A DISTRIBUTED CONTROL SYSTEM FOR LOW-PRESSURE PLANT GROWTH CHAMBERS

A Thesis

by

DENISE LYNN BROWN

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2002

Major Subject: Biological and Agricultural Engineering

A DISTRIBUTED CONTROL SYSTEM FOR LOW-PRESSURE PLANT GROWTH CHAMBERS

A Thesis

by

DENISE LYNN BROWN

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Approved as to style and content by:

December 2002

Major Subject: Biological and Agricultural Engineering

ABSTRACT

A Distributed Control System for Low-Pressure Plant Growth Chambers.

(December 2002)

Denise Lynn Brown, B.S., Texas A&M University

Chair of Advisory Committee: Dr. Ron Lacey

This research focuses on the design and development of a control system for hypobaric plant growth chambers. The chambers will be used to determine the effects of reduced pressure on plant processes and growth. There are two low pressure plant growth systems discussed. The first system consists of two growth chambers that are controlled by a single computer. The control algorithms for measuring and controlling pressure and measuring temperature inside the chambers are written in LabVIEW. This system has no gas concentration measurement or control.

The second system has six growth chambers. The pressure and gas concentrations within each chamber are controlled by PIC16F877 microcontrollers. The microcontrollers also monitor and record temperature. The setpoints for pressure, oxygen concentration, and carbon dioxide concentration are entered into a LabVIEW program on the main computer, and the data is sent to the microcontrollers via serial communication by this same program. Once the microcontrollers have finished adjusting the pressure gas concentrations based on the setpoints and the current conditions, each microcontroller returns values for pressure, temperature, and the amount of each of the three component gases (nitrogen, oxygen, and carbon dioxide) added. These values are read by the LabVIEW program on the computer and stored in files for analysis. The concentrations of the component gases are measured by a process gas chromatograph.

The two-chamber system has limited usage because of a lack of gas control. Tests of the larger system showed that the system can maintain setpoint pressures in the chambers for long durations. The microcontrollers properly adjust gas concentrations using the gas addition algorithm based on setpoints and current concentrations in the chamber. The system will be used to determine the effects of various pressures and gas concentrations on plant processes and development.

ACKNOWLEDGEMENTS

The person that deserves the most acknowledgement, and the most gratitude, is my advisor, Dr. Ron Lacey. You were a source of inspiration when I had none left, an optimist to counter my pessimism, an encyclopedia of knowledge, and a navigator through the "hills and valleys" of this research project. Thank you for everything.

I also would like to thank Andrew Hensley, my friend and the main author of the C code that accomplishes so much of the control algorithm on the Generation III System. Without you, I would never have come to know and love terms such as "pivot gas." I would also probably still be trying to write that code...

I have three semi-hard-working student workers, Garrett Chandler, April Lovelady, and Pei He. All three of you contributed your time, your ideas, and your questions, the last of which helped me most, if only to remind me why I did what I did.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

The future of manned space exploration will rely on life support systems capable of sustaining human activity both onboard spacecraft and on planetary bases with minimum need of resupply from earth. Such a life support system must be able to recycle air and water as well as provide food for a human crew. Higher plants have the capability of recycling both air and water and can provide food as well as psychological benefits for the crew.

Operating spacecraft and planetary bases at reduced pressures reduces the structural requirements as well as the need for resupply of air, water, and nutrients However, the response of plants to reduced pressure is undetermined, and previous experiments show conflicting results. There are few systems in existence capable of reliable, extensive testing of plant growth in reduced pressures. The few systems that do exist are not well documented and cannot be reproduced from available documentation.

There have been three low-pressure plant growth (LPPG) systems at Texas A&M University. The first system, the Generation I LPPG system, was built in the early 1980s, and no documentation exists. The second system is the Generation II LPPG system, which is the upgraded Generation I system. The Generation III LPPG system is a new, larger LPPG system designed to have none of the limitations of the older systems. This research had two goals: first, to quickly redesign and repair the old low-pressure plant growth system that was already at the university in order to begin obtaining plant

---

This thesis follows the style and format of the Journal of Life Support and Biosphere Science.

data, and second, to design a new, modular, and easily expandable LPPG system with more capabilities. Both systems were to be fully documented so that the design would be available to other researchers involved in testing plant growth in hypobaric conditions.

REVIEW OF LITERATURE

Advanced Life Support

Long-duration manned missions will require a life support system capable of sustaining human activity both aboard spacecraft and on planetary surfaces with little or no need for resupply from earth. The physiochemical systems currently employed cannot meet the demands of a long-term mission because of their finite capacity for resource conversion. Thus, the idea of using higher plants in a bioregenerative life support system was born. Higher plants possess the ability to recycle air and water and can provide food as well as psychological benefits for the crew. NASA has been working toward the goal of designing a fully functional system to support the plants and reduce mass costs associated with long duration missions. (2, 21)

According to the NASA-JSC Advanced Life Support System Support Program Plan (2), an advanced life support system must provide the means to recycle resources with minimum need for resupply. Such a system must monitor and control environmental conditions for both the crew and the plants. The conditions for the two might differ on a planetary base, with separate modules for plant production and crew (11). Such a life support system should be self-controlled and require little maintenance to increase crew productivity and safety (2).

Much of the advanced life support research has focused on a Mars planetary base. Martian atmospheric pressure is less than 1/150 that of earth and consists of 95% CO_2. The temperature on the surface of Mars is between –143°C and 17°C. There is virtually no oxygen or water on Mars, unless it is contained within silicon rocks or underground. (10, 16) Erecting a greenhouse on Mars massive enough and airtight enough to withstand the pressure differential between normal earth atmospheric pressure and Mars atmospheric pressure would be prohibitively expensive, difficult to maintain and repair, and difficult to transfer to the Martian surface (3).

Payload considerations also influence the design of a life support system. The cost of moving one pound from the Earth's surface to orbit has been estimated at $10,000. Equivalent system mass is the meter by which various scenarios are compared. Thus, the lighter the materials needed to construct the system, the less of a mass cost accrued by the system. To help reduce mass costs of a life support system and promote ease of construction and maintenance, maintaining the plant growth module of a planetary base at pressures lower than normal earth atmospheric pressure has been considered. This would reduce the mass of a structure needed to withstand the difference between the internal pressure of the structure and the external pressure of the atmosphere of the planet and reduce the amount of gas needed to replace gas lost through leaks and extravehicular activities (EVAs).

Such a plant growth module should be able to maintain a suitable environment for growing plants, which includes a temperature between 15°C and 30°C, providing light levels that will promote plant growth, maintaining a suitable atmosphere with the proper mixture of gases at an acceptable pressure, and having low mass but sufficient volume for growing of plants (10).

Plant Growth Requirements

Plants require light in wavelengths of 400 – 700 nm for photosynthesis. This light is referred to as photosynthetically active radiation (PAR). (20) Plants require irradiation levels of about 400 μmol m^{-2} s^{-1} for C_3 plants and more than 500 μmol m^{-2} s^{-1} for C_4 plants (13).

Plant growth is affected by the concentration of both carbon dioxide and oxygen. How the concentration of CO_2 affects the plant depends on the type of carbon cycle within the plant. C_3 plants carry out photosynthesis using the Calvin cycle, but C_4 plants concentrate carbon dioxide in bundle sheath cells with an additional reaction before carrying out the Calvin cycle. Photosynthesis in C_4 plants is saturated at a partial pressure of 20 Pa CO_2 while photosynthesis in C_3 plants continues to increase at that level. (20)

The partial pressure of oxygen can limit plant growth. The lowest viable partial pressure of O_2 is in the range of 5 to 10 kPa, below which no plant growth can occur (15, 18). Consequently, the lowest practical total pressure at which plants can be grown without modifying the O_2 concentration is 23 to 48 kPa.

Plant growth and development can be hindered by atmospheric contaminants. For example, ethylene is a biologically active plant hormone which can trigger a number of physiological reactions, such as leaf senescence (20), and can be toxic at concentration levels as low as 50 to 100 ppb (22).

Plants also require water and nutrients for growth and development. These, too, must be maintained at acceptable levels in a plant growth module. Plants can be grown either in solid media or hydroponically.

The effects of reduced pressure on plant growth are still largely undetermined, though several experiments analyzing plant response to reduced pressure have been performed. Most of this research has focused on the effects of reduced pressure on the physiological processes of the plants.

Low-Pressure Plant Growth Research

Most low-pressure plant growth experiments have focused on the effects of low pressure on net photosynthesis and dark respiration. There has also been some research conducted to determine germination and growth of plants in simulated Martian atmospheres.

Schwartzkopf and Mancinelli (18) attempted to germinate plants in a simulated Martian atmosphere. Their research showed that plant germination was suppressed in low pressures at elevated levels of CO_2. It also demonstrated the necessity of O_2 for plant germination. From their research, it was concluded that pure Martian atmosphere was not conducive to plant growth. Later studies focus on higher pressure and oxygen levels and lower CO_2 levels.

Corey, et al. (7) used a plant volatiles chamber at Ames Research Center to study CO_2 exchange in lettuce plants at both ambient pressure and 51 kPa. They found increased photosynthesis rates at the reduced pressure and decreased dark respiration rates, though the results seemed to be because of the decrease in oxygen partial pressure occurring at reduced pressure. Carbon dioxide uptake was constant regardless of total pressure.

Daunicht and Brinkjans (8) studied photosynthesis and transpiration rates and morphogenic response of plants at low pressure. They compared photosynthesis rates at three different pressures – 100 kPa, 70 kPa, and 40 kPa - with equal CO_2 concentrations at each pressure and found that photosynthesis rates increased by 2% to 12% at 70 kPa as compared to 100 kPa, but that photosynthesis rates at 40 kPa and 100 kPa were nearly identical. Plant growth parameters such as leaf area and length of stem were adversely affected at the two reduced pressures. Another set of experiments increased the level of CO_2 at the three pressures and found decreased transpiration rate at lower pressures while dry matter production increased at lower pressures. Plant growth parameters responded positively to an enriched CO_2 atmosphere at lower pressures. A final set of experiments compared plant growth parameters at 100 kPa and 40 kPa with a carbon dioxide partial pressure of .04 kPa at both pressures and found that plant growth was inhibited at the lower pressure.

Using the variable pressure growth chamber (VPGC) at Johnson Space Center, Corey, et al. (5) developed a method of measuring leak rates and gas exchange rates at 101.3 kPa and 70 kPa. Photosynthesis and dark respiration were measured at both pressures. They found that the VPGC had a very high leak rate that must be compensated for in future reduced pressure plant research using the VPGC. Leak corrections ranged from 9% down to 3% of the changes in photosynthesis rates over the test period and from 19% down to 4% of the changes in dark respiration rates over the test period. These results allowed the VPGC to be used for plant growth at reduced pressures as long as the high leak rate of the chamber (1.16 to 2.36 chamber volumes/day) was taken into account in measurements of gas exchange rates.

Corey, et al. (6) then studied the effects of reduced pressure and reduced oxygen on photosynthesis and respiration of wheat using the VPGC. Tests were conducted at 101 kPa and 70 kPa. Photosynthesis rates were higher at reduced pressure than at ambient pressure while dark respiration rates were unaffected. Reduced oxygen partial pressure resulted in increased photosynthesis rates regardless of pressure, though the response also depended on the ratio of the partial pressure of oxygen to the partial pressure of carbon dioxide. As this ratio increased, the rate of photosynthesis decreased linearly. Again, dark respiration rates were unaffected.

LPPG Research Systems and Limitations

In spite of the seeming abundance of systems available for studying the effects of low pressure on plant growth, there is little documentation on any of the systems used for the research reported. Also, each system had limitations that severely limited the range of experiments that could be performed in them. There was a definite lack of a fully documented low-pressure plant growth system capable of adequately measuring and controlling pressure and gas concentrations.

Limitations included a lack on environmental control as in the study of germination and growth of wheat in a simulated Martian atmosphere (18). The other systems had similar drawbacks.

The plant volatiles chamber (PVC) used by Corey, et al. (7) had a low leak rate (<1% chamber volume per day). Pressure in the PVC was adjusted manually using a vacuum pump and a valve. Data was collected with an Everex 80286 PC, which was out of date even at the time of the experiment. The data was collected with an Opto 22 Optomux I/O hardware interface, which is a pre-packaged analog or digital I/O unit that acts as a slave to a PC, and controlled using Paragon 500 software. The Optomux hardware was effective, but bulky and not easy to modify. Carbon dioxide concentrations were measured with an infrared gas analyzer. There was no mention of oxygen concentration measurement, nor is the program algorithm given.

The LPPG system used by Daunicht and Brinkjans (8) was an open system that was constantly ventilated with outside air to remove excess CO_2 and ethylene and to maintain a constant oxygen concentration. A mass flow controller and a mechanical precision vacuum controller were used to control pressure, but no specifications were given. Carbon dioxide was added in pulses which were electronically recorded, and concentration of CO_2 was measured using an unspecified infrared gas analyzer. No measurement or control of oxygen concentration was available. No analysis of system performance was given.

The VPGC at Johnson Space Center that was used for research by Corey, et al. (5,6) had excessively high leak rates that required extra tests and compensation equations to determine accurate gas exchange rates. There was no mention of how oxygen concentration measurements were made or what system was used for CO_2 injection to the system. There was also no mention of how pressure was measured and controlled in the VPGC. Because of the high leak rate, the VPGC could not be operated at pressures much lower than 70 kPa.

Schwartzkopf, et al. (17) designed a system of four low atmospheric pressure plant growth chambers. The chambers operated at pressures as low as 1 kPa with a leak rate of 1% of chamber volume per hour. Gases (nitrogen, oxygen, and carbon oxide) were injected using an unspecified on-line gas composition system. Data collection was handled by an Opto 22 interface hardware and an Apple Macintosh II CX computer. The-type of sensors used for pressure, temperature, and gas concentrations were not discussed. There was no analysis of system performance.

Simpson and Young (19) devised a plant growth structure for a Martian derived atmosphere. It could operate at pressures from 133 Pa to 101.3 kPa. There was no control for oxygen and carbon dioxide concentrations, nor was the means of data collection described.

Goto (11) used an environmental control system for plant growth that operated at pressures down to 10 kPa. Gas concentrations were controlled separately, though there was no description of how this is accomplished. The types of sensors were not described. Again, no analysis of system performance was available. The control hardware and algorithms were not mentioned.

In the above system descriptions, there was very little data on system design or performance. Many of the systems that are documented used older technology and were severely limited. Expansion and upgrades would be difficult. Another drawback is that for most of the systems mentioned, there was no ambient pressure chamber, just a low-pressure chamber. The ambient pressure 'control' groups were often placed in open air with no control of conditions whatsoever. Control algorithms were not given for any LPPG system in existence. It was also impossible to perform dynamic experiments in the aforementioned systems (setpoints must remain static during tests).

Data Acquisition and Control in Closed Environment Systems

There have been papers published on the design of systems for monitoring and controlling closed environments. Kacira and Ling (12) discussed the design and development of an automated system for continuous monitoring of plant growth. Most of the paper was dedicated to visual monitoring of plant growth, but there was also some discussion of using a distributed system approach and the necessary sensors and data acquisition hardware. A multifunction data acquisition card was used to read analog signals from two data loggers. However, there were no specifics about the software used to collect data and no circuit diagrams.

Chun and Mitchell (4) reported on dynamic optimization of crop photosynthetic rate in a closed-environment system for crop growth. They used feedback control to adjust the levels of photosynthetic photon flux (PPF) and carbon dioxide concentration. In this way, the optimal amount of PPF and CO_2 for the crop at that stage of growth could be supplied to produce desired results, such as to produce a specific amount of oxygen or to transpire a certain weight of water. Two Minitron II systems were used to maintain the other variables and adjust PPF and CO_2 concentration. Unfortunately, no specifics on this system were published in the paper and no specifications for the Minitron system could be located. (4)

Dynamic control strategies have also been applied to other closed environment systems, such as aquatic systems. Acevedo and Waller (1) developed a model for a simple trophic system containing zooplankton and algae. They then developed a control strategy to maintain a steady state animal population at a certain stage of development by controlling the food supply. The goal was to be able to harvest the animal population at a steady sustainable rate. The models and approaches could also be applied to other closed environment systems such as life support systems.

An optimal control strategy for crop growth in advanced life support systems was developed by Fleisher and Baruh (9). They developed a mathematical model of a feedback control loop to compensate for the effects of environmental disturbances by adjusting PPF. Two crop growth models were considered and two control laws were applied to each model. Several simulations of various conditions were performed to test the control laws. The approach proved to be a potentially useful method of controlling crop growth.

However, it is important to note that most of the dynamic control models have only been tested in simulations. The only system that has dynamic variable control capability is that used by Chun and Mitchell (4). No fully documented low-pressure plant growth system has been developed and used in determining plant response to low-pressure. There is a definite need for a system that allows dynamic control of growth conditions for dynamic testing.

OBJECTIVES

The following objectives were developed to meet the needs for both an immediately operational LPPG system and a new larger, automated LPPG system, both fully documented:

1. Redesign and restore the Generation II LPPG system
    a. Replace and repair the sensors and data acquisition hardware as needed
    b. Develop a control program for the system
    c. Allow for update of setpoints without shutting down the system during experiments
    d. Record real time pressure and temperature data during experiments
2. Design a control system for the Generation III LPPG system
    a. Make a modular system that is easily expanded
    b. Allow for update of setpoints without shutting down the system during experiments
    c. Measure and control pressure and the concentration of oxygen and carbon dioxide in each growth chamber
    d. Record real time pressure, temperature, oxygen concentration, and carbon dioxide concentration data
3. Perform system analysis on Generation III system
    a. Check system components to ensure proper operation
    b. Perform shakedown tests on system

GENERATION II LOW-PRESSURE PLANT GROWTH SYSTEM

System Description

The Generation II LPPG system is an upgrade of the original Generation I LPPG system. No documentation or schematics of this original system exist. When work began, it was discovered that in addition to needing a new control program, most of the sensors and circuitry needed to be replaced, as well as the data acquisition hardware. A description of the system including the modifications follows.

Physical Design

The Generation II LPPG system consists of 2 cylindrical growth chambers, a low-pressure chamber and an ambient-pressure chamber. Both growth chambers consist of a cylindrical body fitted into two flat end plates. The cylinders are housed in a small environmental growth chamber that provides lighting and temperature control. Because there are only two cylinders, results cannot be repeated within an experiment; several repetitions of the same experiment must be run sequentially to produce a statistically sound data set. Figure 1 shows a picture of the system.

The atmosphere in the low-pressure chamber is evacuated with a rotary vane vacuum pump (D5E, Leybold Vakuum GmbH, Cologne, Germany). A complex network of stainless tubing delivers gases to and from the chambers and also carries samples to the gas concentration analyzers. These samples must be pressurized up to ambient pressure and pushed through the analyzers with nitrogen gas. Figure 2 shows a schematic of the system.

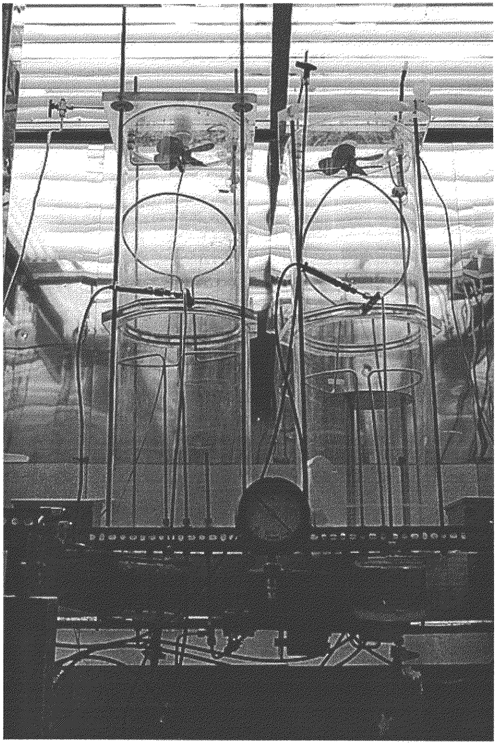


Figure 1. Generation II LPPG System

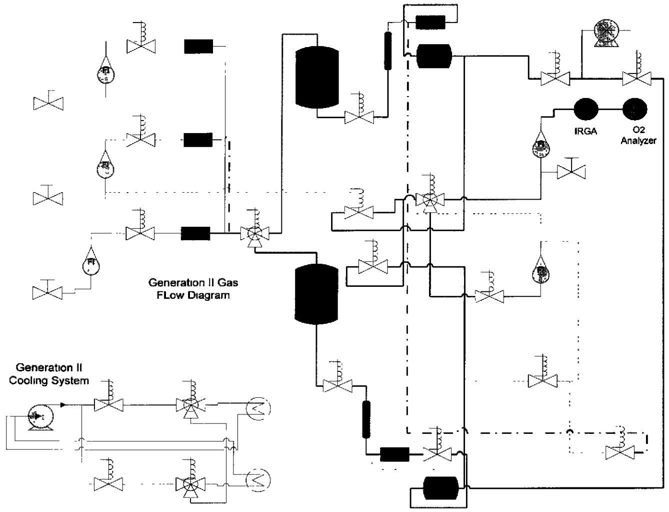


Figure 2. Generation II LPPG System Schematic

Sensors

The pressure sensors from the original system no longer functioned when the upgrades were begun, and the oxygen sensors could not operate at reduced pressure below 70 kPa. Thus it was decided to replace these sensors and to repair the circuits for still-functioning sensors.

Pressure Sensor

Pressure in each chamber is measured by a K-2 pressure transducer from Ashcroft (Dresser Instruments, Addison, TX). The transducers output 4-20 mA in direct proportion to the pressure and have a range of vacuum to 200 kPa .

Temperature Sensor

Temperature is measured using a thermistor in an operational amplifier circuit. The circuit will be discussed below in the analog signal conditioning section.

Infrared Gas Analyzer

In the original LPPG system, carbon dioxide concentration was measured using an 880A infrared gas analyzer (IRGA) from Rosemount Analytical (Orville, OH). The IRGA outputs a signal between 0 and 5 volts that is proportional to the concentration of carbon dioxide gas in the sample. It has a range of 0 to 2000 ppm. The IRGA was to be used in the Generation II system as well. However, the analyzer could never output a reliable or accurate measurement of carbon dioxide even when calibrated. At length, because of time constraints, it was decided to forego measuring carbon dioxide concentration in the Generation II system.

Paramagnetic Oxygen Analyzer

A 775R paramagnetic oxygen analyzer (Rosemount Analytical, Orville, OH) was chosen to replace the original voltaic oxygen sensors. It outputs a voltage between 0 and 5 volts that is proportional to the oxygen concentration in the sample. The analyzer has a range of 0% to 25%. However, like the IRGA, the oxygen analyzer would never output accurate measurements, and so it was decided to forego measuring oxygen concentration, as well.

Mass Flow Controllers

Three mass flow controllers (MFCs) (DFC2600, Aalborg Instruments and Controls, Inc., Baden, Germany) measure and control the flow rate of air, nitrogen, and carbon dioxide into the growth chambers. Each outputs a signal between 0 and 5 volts that is proportional to the flow rate of the gas through the controller. Also, each MFC requires an analog signal from an outside source between 0 and 5 volts to set the valve inside the controller to a certain position. The position of the valve determines the flow rate allowed through the controller.

Computer and Data Acquisition Hardware

Three multifunction data acquisition cards from National Instruments (Austin, TX) were installed in a PC to send and receive signals from the sensors and other hardware. The NI PCI-6503 is a digital I/O card with 24 digital channels that can be configured as either digital inputs or digital outputs. The NI PCI-6023E is the analog input card, which has 16 analog input channels. The NI PCI-6713 has 8 analog output channels. The control program for the entire system is run on a single computer.

Signal Conditioning

Analog Signals

The analog signals in the system consist of the pressure transducer output, the temperature sensor circuit output, the output from the two gas analyzers, and the inputs from and outputs to the MFCs. Some of the temperature sensor and pressure transducer signals require amplification and/or conversion before being input into the appropriate card.

Pressure Sensor

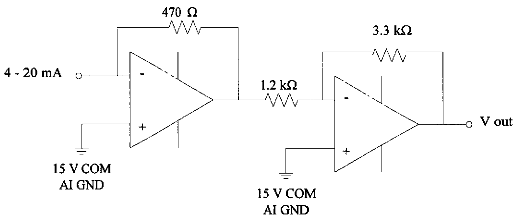
The pressure transducer outputs 4 to 20 mA. However, the data acquisition card can only read voltages between –10 and 10 volts. Thus the output from the transducer must be converted a voltage and then amplified. This is accomplished using the simple operational amplifier circuit shown in Figure 3(a).

Temperature Sensor

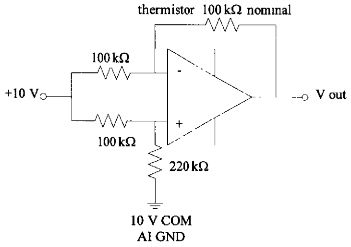
Thermistors are semiconductor devices that change resistance as temperature changes. To measure temperature in each chamber, a thermistor was put into the feedback portion of an op amp circuit supplied with 10 V. The output of the circuit changes as a function of the resistance of the thermistor, which is a function of temperature. Figure 3(b) shows the temperature sensor circuit.

Digital Signals

There are several solenoid valves that must be opened and closed using digital signals, and the vacuum pump that pulls down the pressure in the low-pressure chamber is turned on and off using a digital signal as well. Because the digital I/O card can only



(a)



(b)

Figure 3. Generation II LPPG System Signal Conditioning Circuits. (a) Pressure transducer circuit. (b) Temperature sensor circuit.

supply a 5 V signal at a few milliamps, some means of amplifying the digital signals is needed. Two National Instruments ER-16 electromechanical relay accessory boards provide amplification for the 24 digital signals. These, in turn, actuate larger mechanical relays that can provide the AC power required by the solenoid valves and the vacuum pump.

Control Program

The original program for the Generation I system was written in the C programming language. Variables were hard coded and to change setpoints or other operating parameters, the program had to be shut down, modified, recompiled, and the restarted before changes could take effect in the system. For the new Generation II system program, it was desirous to change setpoints without having to shut down a running experiment. To this end, LabVIEW (National Instruments, Austin, TX) was chosen. LabVIEW is a graphical programming development environment designed for data acquisition and control applications. It easily interfaces with National Instruments data acquisition cards. One of its key features is the ability to update setpoints and other variables without having to stop the control program. Also, LabVIEW comes with a wide variety of instrument drivers in several libraries, which are software routines that carry out specific tasks, anything from adding two numbers to reading analog and digital inputs to complex data analysis. LabVIEW software routines are called virtual instruments, or VIs, and a VI embedded within another VI is referred to as a subVI of that VI.

The control program for the Generation II LPPG system was intended to maintain pressure in each chamber at the setpoint pressure and control oxygen and carbon dioxide concentrations. However, as mentioned, the difficulties in successfully measuring the concentrations of oxygen and carbon dioxide prevented full application of the control program. Still, the system could be used for other tests. To this end, several small subVIs were written. These VIs perform common tasks that are frequently used in the various experiments.

The VIs can be performed in any order desired as long as the correct inputs are supplied. Some of them require a setpoint and/or digital port configuration information. Others require no inputs and only output a value, such as the current temperature. Most of the VIs used in the various control programs require several inputs and outputs, and as long as the inputs are all supplied to the VI, it can carry out its task. The subVIs within a VI are connected to one another using 'wires' that diagram data flow.

Generation II SubVIs

The subVIs are chamber-specific because the pressure and temperature sensors for each chamber have unique calibration equations and because the same type of signal for each chamber has a unique analog or digital channel. The more common subVIs are described below and some of the experimental programs are discussed.

Low Pressure VI

The Low Pressure VI reads the pressure from the pressure transducer on the low-pressure chamber and compares it to the setpoint pressure, which is input by the user in the LabVIEW front panel. If the pressure is greater than the setpoint plus 2.5%, the subVI sends a high signal to the appropriate solenoid valves and turns on the vacuum pump. The program then continually monitors the pressure in the chamber. Once the setpoint is reached, the VI sends out a low signal to the valves and to the pump, thus closing off the valves and shutting the pump off.

If the pressure is less than the setpoint minus 2.5%, the VI sends a high signal to a different set of solenoid valves and opens the valve on the air mass flow controller, allowing pressurized air from a tank to enter the chamber. Again, the VI continually monitors pressure, and when the setpoint is reached, the valves are closed, stopping air-flow into the chamber. Notice there is a 5% dead band about the pressure setpoint. This is to prevent excessive cycling of the vacuum pump.

Ambient Pressure VI

The Ambient Pressure VI performs the same functions for the ambient-pressure chamber that the Low Pressure VI performs for the low-pressure chamber. The only differences between the two programs are the analog input channel read for the sensor, the digital outputs that are changed to open valves, and the calibration equation used to convert the voltage read from the pressure transducer to a pressure. In all other respects, the programs are identical.

Both the pressure VIs record the pressure in the chamber before adjusting it to meet setpoint conditions. It is displayed on the front panel of each subVI and also saved in a file. This data can later be analyzed as necessary.

Temperature VI

This VI reads the voltage from the temperature sensors in each chamber and converts it to a temperature in Celsius. The temperature in each chamber is displayed on the front panel and also recorded in a file.

Single Chamber Flush VI

The Single Chamber Flush VI recycles the air in a single growth chamber. This means that it removes a certain amount of air from the chamber and replaces it with compressed air from a tank. The VI first opens the appropriate solenoid valves and turns on the vacuum pump for a certain amount of time. This time is adjustable in the front panel of the program. The time between flushes is also adjustable. The program records the pressure both before air is pulled out and after the valves have been closed and the pump turned off. These values are used to determine the amount of air that was removed from the chamber.

The Single Chamber Flush VI can be easily modified to flush either chamber. Only the analog input channel read and digital channels for the solenoid valves must be changed and the appropriate calibration input to accomplish this.

There are several other subVIs that perform various functions, such as saving numbers to files, converting between units, and other small tasks, but these are not discussed here.

Generation II VIs

The VIs discussed above are subVIs in the larger programs that operate the Generation II system. Combinations of those and other software routines form the VIs written for the various experiments.

Main VI

The Generation II system has been used to run several tests in an effort to determine the effects of reduced pressure on plant growth. This most commonly used VI simply reads temperature and records and controls pressure in each chamber for the duration of an experiment while other portable equipment is used to measure photosynthesis and other variables of interest in the plant. It is a combination of the Low Pressure, Ambient Pressure, and Temperature VIs.

Two Chamber Flush VI

In the course of the above experiments, which were run for short durations, it was discovered that a plant hormone, ethylene, builds up in the growth chambers and thus a new set of programs was needed to recycle air in the chambers to prevent the buildup of this hormone. The Single Chamber Flush VI was the first program used for this purpose, and was added into the main VI for several experiments to compare ethylene levels between a flushed chamber and a chamber in which air was not recycled.

Eventually, a program was needed to flush both chambers. This VI does that. It contains the Single Chamber Flush subVI, which is set to flush the low-pressure chamber. The low-pressure chamber is flushed first, and the amount of gas removed is

GENERATION III LOW-PRESSURE PLANT GROWTH SYSTEM

System Description

Background

The Generation II system cannot produce replicable experiments in the sense that there is only one control and one experimental chamber. There is no gas concentration measurement or control in the system, thus the effects of gas concentration on plant growth cannot be determined using the older system. Expanding the system to include more growth chambers is not possible because of the small size of the room in which the two cylindrical growth chambers are housed and the integrated nature of the control system – one control program on one computer monitors and controls conditions in both chambers.

Thus the Generation III LPPG system was conceived to be modular, easily expanded, and allow for replicability within each experiment. It was also to have gas concentration control for carbon dioxide and oxygen.

The Generation III LPPG system is located in a large growth room in the Norman Borlaug Center for Southern Crop Improvement. The room provides lighting and temperature control, including photoperiod and light intensity control. There is no temperature control in the system other than that supplied by the growth room.

Physical Design

The Generation III LPPG system has six cylindrical growth chambers. Each chamber has a volume of 55.3 liters. Gases and water are added and removed via five vacuum feedthroughs. The temperature sensor signal passes out of the chamber through calculated. Then the same molar percentage of gas is removed from the ambient-pressure chamber.

Discussion

The LabVIEW control programs can control and monitor conditions in both cylindrical growth chambers. Using LabVIEW allows for real time update of setpoints during system operation. The control programs for each experiment can be easily modified and adjusted using the set of subVIs already written, which would allow a large variety of experiments to be run.

The Generation II LPPG system is adequate for short experiments to determine the effect of reduced pressure on plants as long as each chamber is flushed and supplied with fresh air to eliminate the effect of gas concentrations on plant growth and external equipment is used to measure photosynthesis rate, transpiration rate, etc. However, the lack of gas concentration measurement and control prevents the system from being useful in a full range of tests to determine the effects of reduced pressure on plant growth and limits the types of experiments that can be run. These limitations are eliminated in the Generation III Low-pressure Plant Growth System.

a vacuum-rated electrical feedthrough. The chambers are mounted in pairs on wire racks and all of the power supplies, sensors, and circuitry associated with each chamber are mounted on the bottom of that rack. The six chambers share a rotary vane vacuum pump (BD-20A, Bestech, North Bergen, NJ), which is used to evacuate gases from each of the six chambers. Figure 4 shows a photograph of the Generation III LPPG system. A schematic of the system is shown in Figure 5. For a more detailed description of the design of the growth chambers, refer to 'Engineering Design of a Hypobaric Plant Growth Chamber' (14).

Sensors

Pressure Sensor

The pressure inside each chamber ranges from a slightly above vacuum to atmospheric pressure, which is 101.3 kPa. Thus a transducer that would produce a linear output over that range is desirable. The sensor needs to output a voltage between 0 and 5 volts, which is within the range of many data acquisition devices. Thus an Ashcroft K-2 pressure transducer (Dresser Instruments, Addison, TX) with a range of 0 to 101.3 kPa and an output range of 1 to 5 volts was chosen.

Temperature Sensor

The thermistor circuit in the Generation II system measures temperature accurately, but for the Generation III system, a different approach has been taken. LM35DT precision centigrade sensors that output 10 mV/°C are used. (National Semiconductor Corporation, Santa Clara, CA) The sensor's output is highly unstable unless the output and ground are tied together in a voltage divider circuit recommended

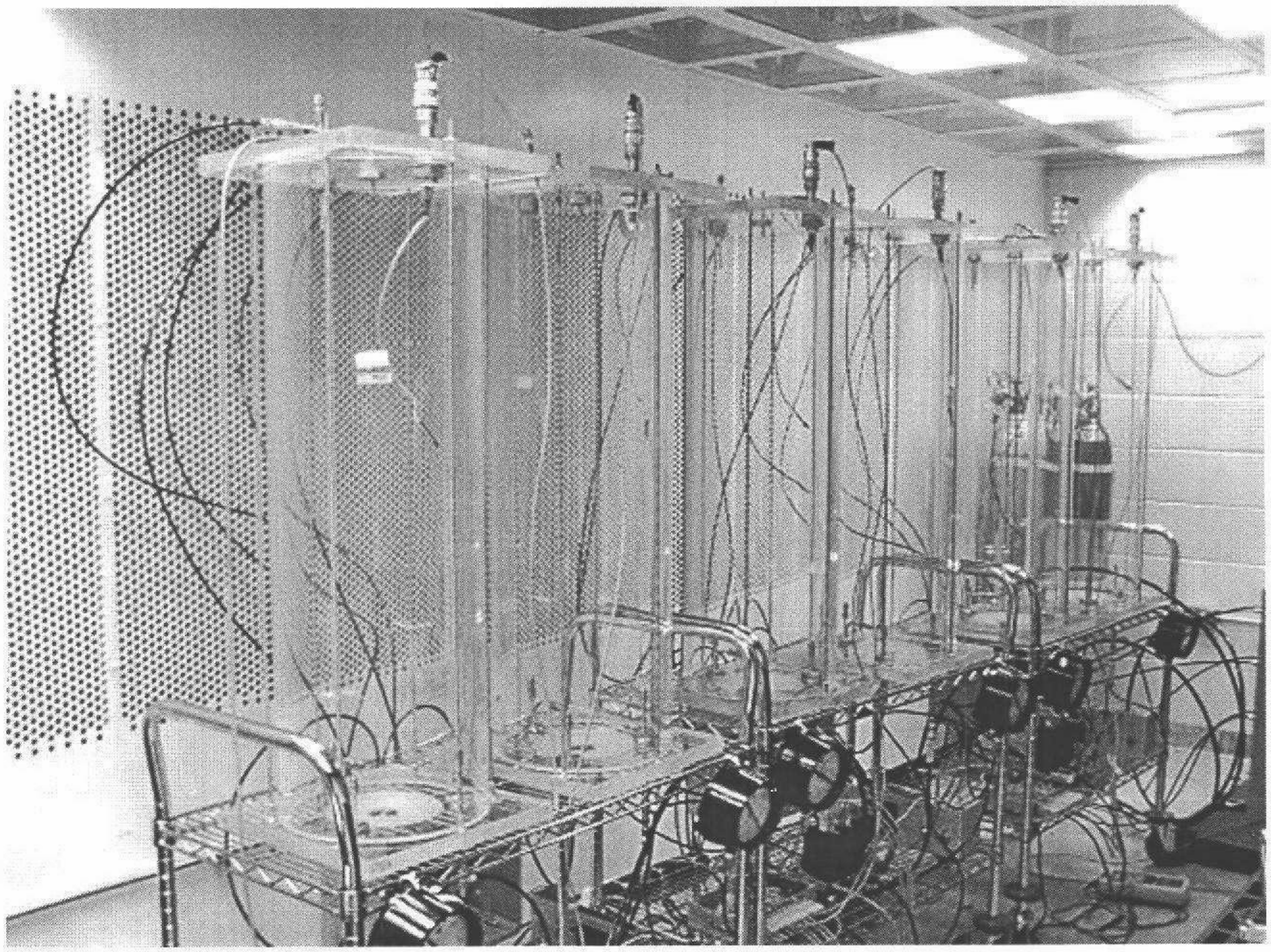


Figure 4. Generation III LPPG System

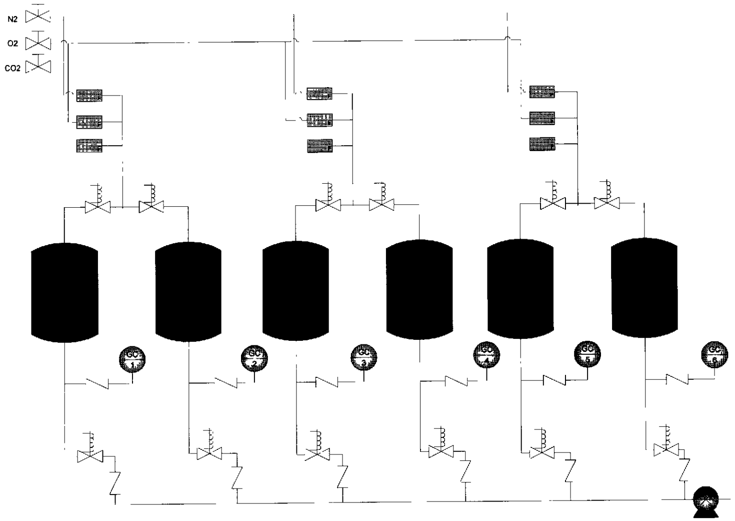


Figure 5. Generation III LPPG System Schematic

by the manufacturer. The output must also be amplified, as it is too small to accurately read with most data acquisition devices. To give the sensors more thermal mass, they are mounted on the metal plant stands in the chambers using thermal tape. The result is a highly stable, accurate measurement of temperature in each chamber.

Mass Flow Controllers

The environment inside each chamber will consist of three gases: nitrogen, oxygen, and carbon dioxide. These gases are added into the chambers in their pure form from three gas tanks. To this end, there are three MFCs for each set of two chambers. (1179, MKS Instruments, Andover, MA) One is calibrated for nitrogen, one for oxygen, and one for carbon dioxide. The MFCs have an accuracy of 1% FS. The MKS MFCs were chosen because of the wide range of flow rates available and because of their accuracy.

Each MFC outputs a signal between 0 and 5 volts that is proportional to the flow rate of gas through that MFC. And like the Aalborg controllers, an analog signal between 0 and 5 volts must be input to the controller to set the desired flow rate.

Gas Chromatograph

Because of the problems with the IRGA and the paramagnetic oxygen analyzer, a new approach to measuring gas concentrations in the chambers was needed. Gas chromatography has been used in many fields to measure the concentration of organic compounds. The theory can also be applied to a variety of gases. Thus a process gas chromatograph was designed specifically for the low-pressure application. (GCX, Rosemount Analytical, Orville, OH) The GCX measures the concentration of carbon dioxide, oxygen, and nitrogen in each chamber.

Gas Chromatography

Chromatography refers to a group of methods that involve transport of a sample through a column. The column contains a partitioning agent that can be either solid or liquid supported by a solid. This partitioning agent is the stationary phase. The sample can be either a liquid or a gas and is referred as the moving phase. The transport of the sample through the column results in separation of the components of the sample because of the selective retention exerted by the portioning agent. Light molecules travel through the column more quickly than heavier molecules. The different components segregate into separate bands at the exit of the column. There the individual bands are directed to a detector to determine the relative concentration of each component, and the time it takes the component to travel through the stationary phase is used to identify the component. The GCX uses a thermal conductivity detector.

GCX Operation

The GCX has six sample streams - one for each chamber - and one calibration stream. It analyzes the six streams in order and stores the concentrations of oxygen, nitrogen, and carbon dioxide in both a chromatogram and in the modbus. (Modicon 584, Schneider Electric, Paris, France) The seventh stream has also been configured as a "dummy" stream that is used for timing purposes. The dummy stream brings in air from a compressor, but no data from that stream is used except when stream seven is configured as the calibration stream. A manual valve selects the stream source for the 7th stream.

All GCX setup and configuration functions as well as calibrations are handled by the GCX maintenance software. This software interfaces with the GCX and allows access to GCX status, chromatograms, analysis results and chromatograms, and is used to program operation parameters into the GCX.

The process GCX designed for use in the Generation III LPPG system can analyze samples at pressures ranging from 25 kPa to atmospheric pressure. The full-scale range of measurement varies for each component. The GCX can measure up to 100% nitrogen, 30% oxygen, and 1% carbon dioxide.

Modbus Protocol

The GCX normally outputs a chromatogram and gas concentrations to the computer using the human-machine interface software provided by the manufacturer. However, this software does not allow the concentration values to be read and stored in an individual file for each concentration. Thus, the gas concentrations must be read from the GCX using Modbus Protocol. Modbus Protocol is a messaging structure used to establish master-slave communication between intelligent devices. Commands are sent from the computer, which is the master device, to the GCX, which is the slave device, in a specific format, and the GCX returns information based on that command. The hardware in a device that communicates using this protocol is usually referred to as a modbus. The modbus in the gas chromatograph is a Modicon 584. (Schneider Automation, Inc. Paris, France)

The GCX modbus communicates using half-duplex RS-422 carried on full-duplex wiring. The signal is converted to RS-232 and input into a serial port on the computer (PC). Commands are also sent from the PC to the GCX using this serial port. The modbus commands can be in one of two formats – referred to as framing – ASCII and RTU. The type of framing dictates the format of the data sent to and from the modbus. The default framing for the GCX is RTU.

As mentioned above, Modbus Protocol is a messaging structure, and the framing is part of that structure. In a way, the protocol resembles a very simple programming language. Commands are in the form of numerical function codes. Each function code tells the modbus to perform a specific action. The most commonly used function codes request data from the slave device. Data is stored in registers in the slave device. Some different function codes refer to different data registers. The GCX uses an abbreviated version of Modbus protocol – not all commands are available. The function codes that will be used on this project simply request that the modbus return data from specific registers. Table 1 shows the available function codes.

Each sample stream on the GCX has a corresponding counter register which increments by one every time the GCX finishes a sample for that stream. (Stream numbers correspond to chamber numbers: stream 1 is the sample from chamber 1, and so on.) The computer polls this register (function code 03), and if it has changed from its previous value, the computer then asks the GCX to send the new concentrations for that stream (function code 04), which are stored in 3 other registers – one register per

Table 1. GCX Modbus Function Codes. The four function codes allowed on the GCX are listed below along with a description of the commands.

Function Code	Name	GCX Function Description
02	Read Digital Registers	Used to read digital status from GCX
03	Read Output Registers	Used to read GCX status information stored in 16-bit registers. This register contains the stream data counters
04	Read Input Registers	Used to read GCX analysis results, which is stored in 16-bit registers
08	Loop Back Test	Used for diagnostic purposes; returns the contents of the received message.

component gas. The counters for each stream are stored in holding registers. The gas concentrations are stored in input registers.

Once the computer receives the data from the GCX input registers, it must convert those numbers to a useable form. The numbers stored in the modbus are integers from 0 to 4095 that represent a percentage of the full-scale concentration for that gas.

For example, if, for oxygen concentration of chamber 1 the modbus returns the value 678, this represents an oxygen concentration between 0 and 30%. Thus 678 is equal to 4.97%. An example is shown below.

$$\frac{\%\_O_2}{30\%} = \frac{N}{4095}$$

$$\frac{\%\_O_2}{30\%} = \frac{678}{4095}$$

$$\%\_O_2 = \frac{678 * 30\%}{4095} = 4.97\%$$

Carbon dioxide and nitrogen concentration are calculated in a similar manner. Carbon dioxide has a range of 0 to 1% and nitrogen can be between 0 and 100%.

Tables 2 and 3 show an example of a modbus command sent to the GCX to return the oxygen concentration for chamber 1 and the reply.

Signal Conditioning

There is very little signal conditioning required for most of the sensors. Every analog input in the system passes through a low-pass filter, which eliminates transient voltage spikes. The pressure transducer and MFC outputs do not require further

Table 2. Example of a Modbus Command. The command includes the device address, the function code, the starting address of the register to be read, the number of registers to be read, and an error code.

Address	Function Code	Address of Starting Register	Number of Registers to Read	CRC
\$01	\$04	\$01	\$01	\$XXXX

Table 3. Example of Modbus Reply. The Modbus reply includes the device address, the function code, the data requested, and an error code.

Address	Function Code	Register Data	CRC
\$01	\$04	14	\$XXXX

adjustment before being read. However, the temperature sensor and the digital signals to the vacuum pump do require some amplification. Each analog signal passes through a low pass filter before being input into the microcontroller to filter out transient voltage spikes. Diagrams and schematics of the pressure transducer and mass flow controller signals are shown in Appendix B.

Temperature Sensor

As mentioned above, the temperature sensor signal is not stable unless it is part of a voltage divider circuit. The circuit used in this system is one of those recommended by the manufacturer. The sensor output must also be amplified. This is accomplished using a simple non-inverting operational amplifier circuit. The op amp is an LM358 (National Semiconductor, Santa Clara, CA). Figure 6 shows the temperature sensor circuit and the sensor range available with that circuit.

Vacuum Pump Signal

The vacuum pump requires 15 amps to run, and the inrush current is many times greater. Operating such a large pump with a small TTL signal requires a large relay capable of supplying a lot of current. To prevent the high power levels from harming the data acquisition and control devices, two relays are used in series. The first relay is a small optoisolated relay (AQV251, NAiS, Osaka, Japan). It isolates the digital signal used to turn the pump on and off from the AC signal necessary to operate the pump. The relay requires an input between 3 and 32 V DC for a high signal and outputs 5 V, and at low, it outputs zero, or ground. The relay isolates the control signal from the pump.

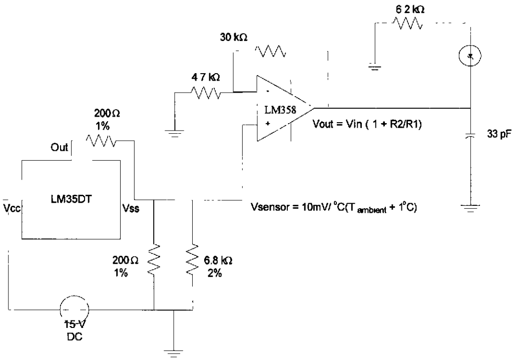


Figure 6. Temperature Sensor Circuit.

The second relay that actually operates the pump is a solid-state relay (D2450, Crydom, SanDiego, CA). This relay takes the signal from the small relay and converts it into an AC signal with enough current to activate the pump. The relay can supply up to 50 amps. Because of the high operating current of the pump, the relay must be mounted on a massive heat sink. A schematic of the circuit is included in Appendix B.

Additional Hardware

Experiments in the Generation II system showed that condensation builds up inside the growth chambers. A large refrigeration unit (JT500, Koolant Koolers, Inc., Kalamazoon, MI) and a cooling coil are used to remove excess water from the chambers and condense water vapor out of the humid atmosphere within the cylinders. The condensation removal loop is coupled with an ethylene filter. Currently, potassium permanganate is used to remove excess ethylene from within the growth chambers. This loop is under manual control, but in the future, it will be automated.

Microcontroller

The Generation II system has as single computer controlling both growth chambers. This approach was not practical for a larger system with more variables and more chambers. Thus, a means of controlling each chamber individually in a distributed control system was necessary. Each chamber could be controlled using a microcontroller. That microcontroller (MCU) monitors and controls pressure and gas concentration levels in each chamber and records the temperature in the chamber.

A microcontroller to control all of the conditions in a chamber to satisfaction required five analog input channels: pressure, temperature, oxygen mass flow rate, carbon dioxide mass flow rate, and nitrogen mass flow rate. It also required at least 3 digital outputs for activating solenoid valves and operating the vacuum pump and one digital input. The digital input was needed to read a pin on the MCU that shares the MFCs to determine if the MFCs were already in use. A means of communicating with the GCX and with a computer was necessary, thus the MCU needed to be capable of serial communication. The MFCs each require an analog signal between 0 and 5 volts to set the flow rate through the controller. Microcontrollers cannot output a true analog signal except at 5 volts, but a pulse width modulation (PWM) signal can emulate an analog signal, so a microcontroller with three PWM channels was desirable. The program necessary to control conditions in the chamber was complex, so adequate programming space was needed. Because the gas concentration calculations require many equations and good accuracy, a compiler with floating-point math capability was needed. For ease of programming, a compiler that uses a high level programming language was desirable.

The PIC16F877 MCU ((Microchip Technology, Inc., Chandler, AZ) met most of these requirements, and had other benefits as well. It has eight 10-bit A/D converters, 33 digital I/O channels, RS-232 capabilities, 2 PWM channels, and 8k of program space. The chip also has flash program memory, which means it can be programmed multiple times. A third party C compiler from Custom Computer Services (Brookfield, WI) allows floating point math and has several useful built in commands for serial communication and A/D conversion. A schematic of the chip including channel assignments and the serial conversion chip is included in Appendix B.

Computers

There are two computers in the control system for the Generation III system. One of them is devoted wholly to running the GCX-HMI maintenance software. This was necessary because any other actions at the computer communicating with the GCX will interrupt communication between the PC and the GCX. This computer is used to check GCX diagnostics while it is running, and to perform the manual calibrations of the GCX.

The other computer acts as a switchboard and control panel. It directs data to and from the microcontrollers and reads the component concentrations for the GCX modbus. Programming the pressure and gas concentration setpoints into the microcontrollers would mean that to change those setpoints, the microcontrollers must be removed from the system, reprogrammed with the new setpoints, and then reinserted into the system. However, as in the Generation II system, real time update of setpoints is desirable to perform dynamic testing of conditions in the growth chambers. Thus setpoints are input by a user into the main PC and sent to the microcontrollers via serial communication.

Device Network

There are several devices involved in the control system for the six growth chambers. These devices do not all need to communicate with each other, but they do all communicate with the main PC. Figure 7 shows the communication network between devices. The main computer sends setpoints to each of the six microcontrollers via RS-232 serial protocol. It also receives pressure and temperature data the same way. The

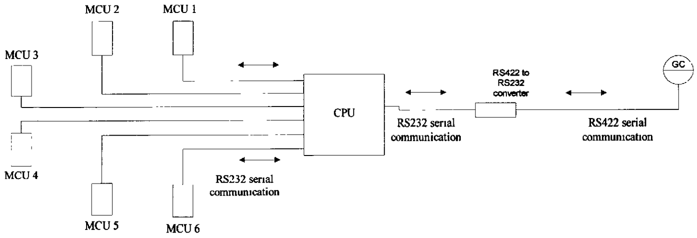

MCU 2
MCU 1
MCU 3
MCU 4
MCU 5
MCU 6
CPU
RS232 serial communication
RS232 serial communication
RS422 to RS232 converter
RS422 serial communication
GC

Figure 7. Device Communication Network

computer communicates with the GCX modbus using the same serial protocol. The RS-422 signal from the GCX is converted to RS-232 using a 485OT9L serial converter (B&B Electronics, Ottowa, IL).

There are seven serial ports needed for all of the serial communication between devices. These serial ports are provided by an NI PCI-232/16 serial expansion card (National Instruments, Austin, TX). The card and the included breakout box have 16 available serial ports, and it requires only one interrupt address from the PC to run all 16 ports.

Software

There are three different sets of software used in the control system for the growth chambers. As mentioned before, the GCX has maintenance software, the GCX-HMI software. Special software was used to compile and load the MCU programs. The main computer runs several LabVIEW programs to operate the entire LPPG system and pass data between the MCUs and the GCX.

Microcontroller Software

The PIC MCUs are programmed in an integrated development environment (IDE) known as MPLAB. (Microchip Technology, Inc., Chandler, AZ) MPLAB has an assembly compiler, a simulator, debugger, and other features. It also interfaces with the PICSTART PLUS programmer (Microchip Technology, Inc., Chandler, AZ) that is used to program the chips. The MCU operating program is written in C. The third party compiler used to compile the C code for the chips also works within MPLAB.

Program Overview

The program for each microcontroller is responsible for maintaining setpoint conditions within its associated chamber. Because the MCUs have limited storage space, pressure, temperature, and volume of gas added to each chamber are returned to the computer at the end of each program loop.

None of the three controlled variables are independent variables. Each is linked to the other two controlled variables and to chamber temperature as well. Thus there is a single control algorithm for all three controlled variables – pressure, carbon dioxide concentration, and oxygen concentration.

Program Algorithm

The program for each chamber is identical to that of the other chambers except for the calibration equations for the pressure transducer and temperature sensor that are particular to that chamber. The control algorithm is an infinitely repeating loop that first receives the setpoints and gas concentrations from the main computer and then adjusts the pressure and gas concentrations to reach those setpoints.

To do this, the MCU first reads the voltage from the pressure transducer, converts it to a pressure in kilopascals, and compares that value to the pressure setpoint. If the setpoint pressure is much lower than the current pressure, the MCU sends out two high digital signals to turn on the vacuum pump and open a solenoid valve, allowing the pump to draw down pressure in the chamber. When the setpoint is reached, the pump is shut off and the valve closed.

If the pressure is equal to or lower than the setpoint, the MCU proceeds to the gas concentration section of the program. It reads the voltage from the temperature sensor and converts that value to a temperature in degrees Celsius. It then calculates the total number of mols of gas in the chamber using the ideal gas law. The values of pressure and temperature used in the calculation are those read from the sensors on the chamber.

$$n\_gas\_total = \frac{PV}{RT}$$

The mols of each component gas present in the chamber is then calculated based on the concentration of that gas returned by the gas chromatograph, which is in percent form.

$$n\_gas[i] = \frac{[gas[i]]}{100} n\_gas\_total$$

The MCU then calculates the number of mols of gas needed in the chamber to reach the setpoint pressure. This is also obtained using the ideal gas law, with the setpoint pressure as an input.

$$n\_gas\_set = \frac{P_{set}V}{RT}$$

The next step is to determine the 'pivot gas'. If the number of mols of each gas currently in the chamber is assumed to be the number of mols of that gas present at the setpoint concentration of that gas, a value for the total number of mols of gas in the chamber necessary for that to be the case is obtained. This is done for each of the three gases.

$$n\_gas\_set[i] = \frac{n\_gas[i]}{\left(\frac{[gas[i]]_{set}}{100}\right)}$$

The gas that requires the greatest number of mols of total gas in the chamber is the 'pivot gas' and the other 2 gases are added to balance the concentrations in the appropriate proportions. If the total number of mols of gas required in the chamber to reach the pressure setpoint (*n_gas_set*) is greater than that needed for any of the 3 gases to reach setpoint concentration (*n_gas_set[i]*), then there is no pivot gas per se and all three gases are added. For clarification, please refer to Figure 8, which is a flow diagram of the control algorithm.

Once the pivot gas is determined, the number of mols of each gas to be added to the chamber is calculated and then converted to a volume using the molecular weight and density of that gas at standard temperature and pressure.

$$n\_add[i] = \frac{[gas[i]]_{set}}{100} n\_gas\_set[pivot\_gas] - n\_gas[i]$$
$$v\_add[i] = \frac{n\_add[i]MW[i]}{\rho[i]}$$

The proper amount of each gas is added one at a time through the mass flow controllers, which measure mass flow in standard cubic centimeters per minute. A digital signal opens the gas inlet valve to the chamber and a PWM signal is sent to each mass flow controller in turn. This signal tells the control valve in the MFC how far to open and controls the flow rate of gas into the chamber.

Each MFC sends an analog signal back to the MCU that is directly proportional to the flow rate of gas through that MFC. A trapezoidal integration over time converts the flow rate provided by this signal into a volume added. When the correct amount of each gas has been added, the PWM signal to its MFC is shut off and the next MFC is

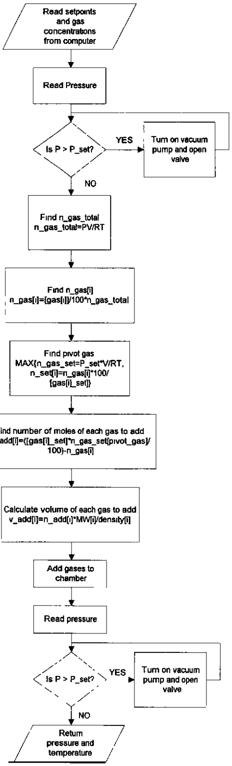

Read setpoints and gas concentrations from computer
Read Pressure
Is P > P_set?
YES
Turn on vacuum pump and open valve
NO
Find n_gas_total n_gas_total=PV/RT
Find n_gas[i] n_gas[i]=[gas[i]]/100*n_gas_total
Find pivot gas MAX{n_gas_set=P_set*V/RT, n_set[i]=n_gas[i]*100/[gas[i]_set]}
Find number of moles of each gas to add n_add[i]=([gas[i]_set]*n_gas_set[pivot_gas]/100)-n_gas[i]
Calculate volume of each gas to add v_add[i]=n_add[i]*MW[i]/density[i]
Add gases to chamber
Read pressure
Is P > P_set?
YES
Turn on vacuum pump and open valve
NO
Return pressure and temperature

Figure 8. MCU Control Program Algorithm

engaged until all three (or two) gases have been added. When all three gases have been added, the valve is closed. The MCU then again checks pressure and draws the pressure back down to setpoint if it is too high. After all adjustments are made, the MCU sends the final pressure and temperature back to the computer for storage in a file.

Each microcontroller carries out the control algorithm independently of any other microcontroller. There is no communication between MCUs except during the addition of gases. Each set of two chambers has but one set of mass flow controllers. Thus the MCUs turn a digital pin 'high' while adding gas, and the other MCU on that rack waits to add gas if the corresponding pin on its partner is high. In this way, an accurate amount of each gas is added to each chamber without interference from the microcontroller of the chamber sharing the mass flow controllers.

The C code used to implement the control algorithm is given in Appendix A.

LabVIEW Programs

The LabVIEW programs for the Generation III system are concerned mainly with communication. The MCU programs send data to the MCUs and receive data in return. The GCX program communicates with the modbus on the GCX. It sends commands to the modbus requesting specific data and waits for a response.

Serial communication in LabVIEW can be accomplished in two ways: by using the basic serial commands or by using LabVIEW Virtual Instrumentation Software Architecture (VISA) commands. The MCU communication programs use VISA. VISA is a set of software routines for configuring, programming, and troubleshooting instrumentation systems comprised of VXI, VME, PXI, GPIB, and/or serial interfaces.

Communication with the GCX modbus could also be achieved using VISA commands, but it would require extensive formatting of the data and special attention to timing. Instead, the PC communicates with the modbus using BusVIEW, which is software designed to communicate with industrial controllers. (Software Engineering Group, Wayland, MA) Installing BusVIEW adds a set of software routines for communicating with the modbus to the LabVIEW virtual instrument library. To use these VIs to communicate with the GCX, the correct serial settings, the type of framing desired, and the address of the GCX must be setup in the BusVIEW Control Panel.

Overview

There are six VIs that communicate with the six MCUs. Each VI is specific to a certain growth chamber and named accordingly. There is one GCX communication VI. As seen in the communication network diagram, the PC is the central switchboard that relays data between the GCX and the MCUs and stores data from each for analysis.

An important part of this data relay relies on global variables. Global variables are variables that can be accessed by several different VIs running at the same time. The concentrations of nitrogen, oxygen, and carbon dioxide in each chamber are global variables. The GCX VI writes to these variables, and the various MCU programs read the value of the appropriate global variables when sending data to the MCUs.

MCU/Chamber VIs

Each chamber has a LabVIEW program that sends setpoints and gas concentration values to its MCU and reads the data from that MCU at the end of each program loop. There are two main communication subVIs in each chamber VI, the Send Data VI and the Receive Data VI.

Send Data VI

The Send Data VI sends the pressure setpoint, oxygen concentration setpoint, carbon dioxide setpoint, and the gas concentrations to the MCUs via RS-232 serial communication. Because the MCUs cannot easily convert a string of characters representing a floating point number into an actual number for use in calculations, the oxygen setpoint and the oxygen and nitrogen concentrations are multiplied by a factor of 10 and the carbon dioxide concentration is multiplied by 1000. After converting the concentrations, the VI uses VISA commands to send the setpoints and the latest gas concentration variables from the GCX to the MCU. Figure 9 shows a picture of the front panel of a chamber VI and Figure 10 shows the program. The only difference between the programs for the various chambers is the string used to identify the serial port used. The correct string for the serial port for the appropriate chamber is selected in the VISA Resource subVI. This subVI has one input (the chamber number) and outputs the correct serial port address string.

Read Data VI

The Read Data VI waits for data from the MCU. It reads the pressure, temperature, and amount of each gas added from the appropriate serial port and saves that data in files.

There were some difficulties in reading data from the microcontrollers. The VISA Read VI is supposed to read until it gets the number of character specified, the

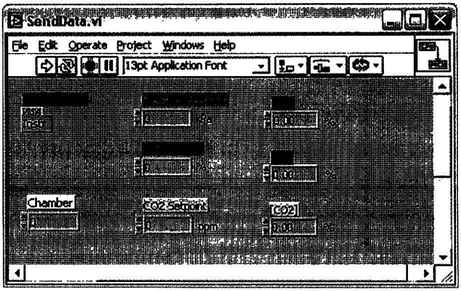


Figure 9. Send Data VI Front Panel

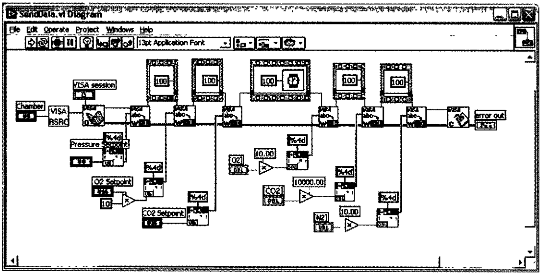


Figure 10. Send Data VI Program Diagram

timeout value is reached, or a termination character is received. However, the VI does not wait for these conditions before terminating the read operation. Thus, the Number of Bytes Available at Serial Port setting in the VISA property node was included in the program along with a short time delay and the property node and read operation are placed in a loop that repeats until data is received. The VISA Read VI is accessed only after the program has detected data at the serial port. At that point, the VI reads all of the characters from the port. Because the read operation can occur while the MCU is still sending data, the serial port is read again, and then all of the data is then sorted and stored in the appropriate files using the File subVI. Figure 11 shows the front panel of the Read Data VI and the program is shown in Figure 12.

Chamber VIs

The six chamber VIs contain an infinite loop that first reads the current gas concentrations from the appropriate global variables for the chamber and then sends that data and the user-input setpoints using the Send Data VI. The VI then reads data back from the MCU using the Receive Data VI. Figure 13 shows the front panel for the Chamber 1 VI. There is a ten minute minimum time between loops.

GCX VI

There is one main GCX VI, called Save GCX Data, but in it are several smaller subVIs that carry out essential tasks, such as reading certain registers in the GCX modbus. The first loop of the Save GCX Data VI sets the global variables that contain the gas concentrations for each chamber to the default values of 21% oxygen, 1000 ppm carbon dioxide, and 78.9% nitrogen. The program then waits for half an hour before

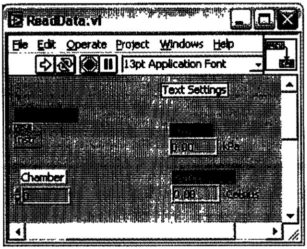


Figure 11. Receive Data VI Front Panel

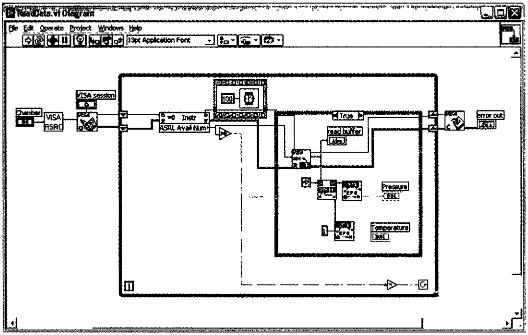


Figure 12. Receive Data VI Program Diagram

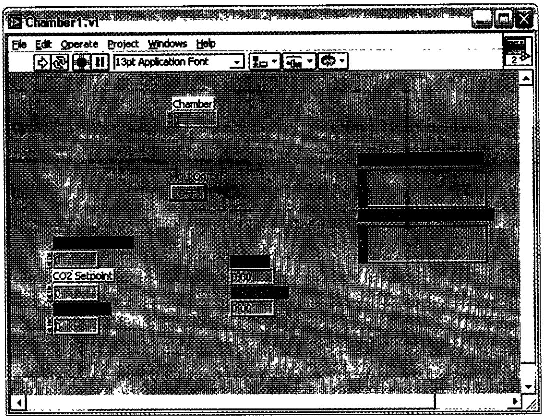


Figure 13. Chamber 1 VI Front Panel

polling the GCX modbus for the stream 1 component concentrations. This is accomplished by reading the appropriate modbus channel using the Read Input Register VI. The Save GCX Data VI repeats this process for each of the sample streams in an infinite loop. The loop repeats every 30 minutes. For a flow diagram of the Save GCX Data VI program algorithm, refer to Figure 14.

Because all of the communication with the GCX is through a single serial port, multiple programs cannot be used to read GCX data. LabVIEW programs take sole possession of a serial port and do not allow any other application, including another LabVIEW program, to access the serial port. The front panel of the Save GCX Data VI is shown in Figure 15.

System Performance

Once the Generation III LPPG system construction was completed, several tests to check that the various system components functioned correctly were run. The first step was to calibrate the sensors.

Sensor Calibrations

Both the pressure and temperature sensors required calibration. Small C programs were loaded onto the MCUs to read the sensors and return data to the PC.

Pressure

Two C programs were written for the pressure calibration. One program took pressure readings while evacuating the chamber, and the other took pressure readings while increasing pressure in the chamber. Both programs returned the 10-bit number

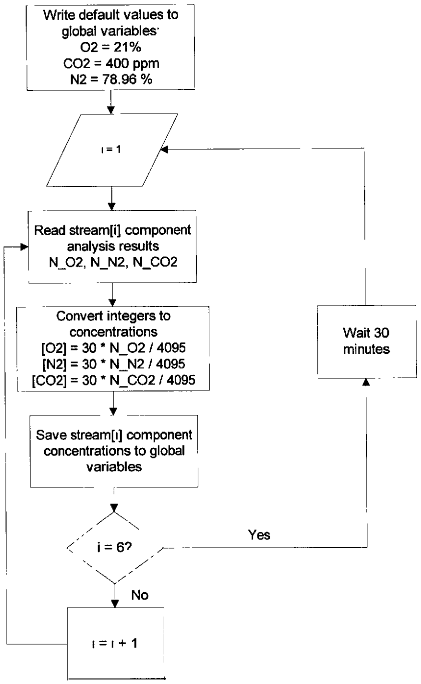

Write default values to global variables:
O2 = 21%
CO2 = 400 ppm
N2 = 78.96 %
i = 1
Read stream[i] component analysis results
N_O2, N_N2, N_CO2
Convert integers to concentrations
[O2] = 30 * N_O2 / 4095
[N2] = 30 * N_N2 / 4095
[CO2] = 30 * N_CO2 / 4095
Save stream[i] component concentrations to global variables
i = 6?
Yes
No
i = i + 1
Wait 30 minutes

Figure 14. Save GCX Data VI Program Algorithm

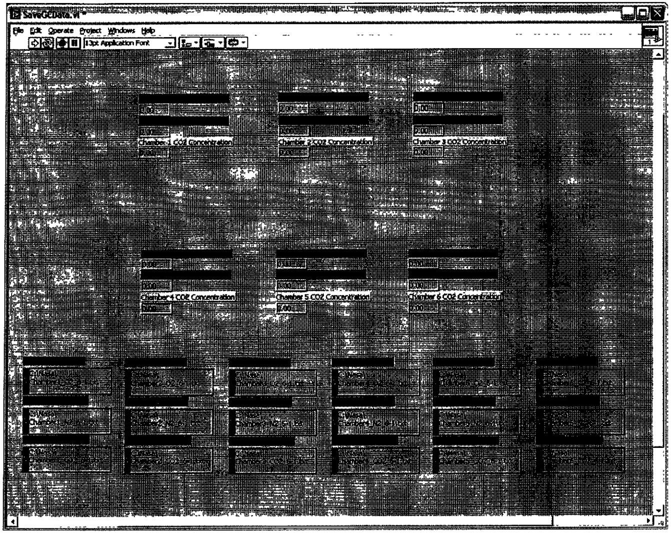

SaveGCData.vi
File Edit Operate Project Windows Help
13pt Application Font
Chamber 1 CO2 Concentration
Chamber 2 CO2 Concentration
Chamber 3 CO2 Concentration
Chamber 4 CO2 Concentration
Chamber 5 CO2 Concentration
Chamber 6 CO2 Concentration

Figure 15. Save GCX Data VI Front Panel

output by the A/D converter to the PC, where the values were stored in a file. In this way, variations in the A/D converter on the MCU could be taken into account. The two sets of data were compared to manual pressure readings from a calibrated vacuum gauge (Dresser Instruments, Addison, TX) taken as the programs were running. To eliminate error from the pressure transducer signal settling time, a small time delay between changing the pressure in the chamber and reading the pressure was built into both programs. The pressure was stepped up or stepped down in small increments during both tests. The procedure was repeated twice for each chamber and as both sets of data were nearly identical, no more repetitions were deemed necessary.

The two calibration programs were necessary to determine if there was hysteresis in the data. Only chambers 2 and 4 showed any indication of possible hysteresis, but at this time, there were problems identified with heating in the chamber 4 MCU that were later eliminated. This eliminated the variation in the pressure transducer readings for chamber 4. At a later date, a statistical analysis on the data for chamber 2 will be performed to determine if there are hysteresis effects and these will be accounted for is necessary. The heating had caused the A/D output of the chip to increase with time. Figures 16-21 show the calibration results for the pressure transducers on the six chambers including the calibration equations. The C code for the pressure calibrations is given in Appendix A.

Temperature

The temperature sensors, while pre-calibrated to output 10 mV/°C, required calibration. This was due both to the stabilizing circuit offset and range limitation, and

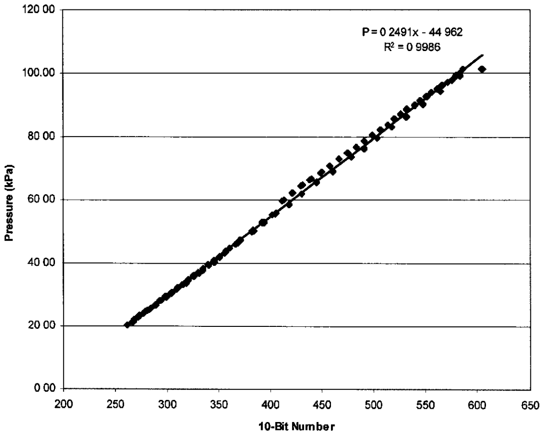


Figure 17. Chamber 2 Pressure Transducer Calibration

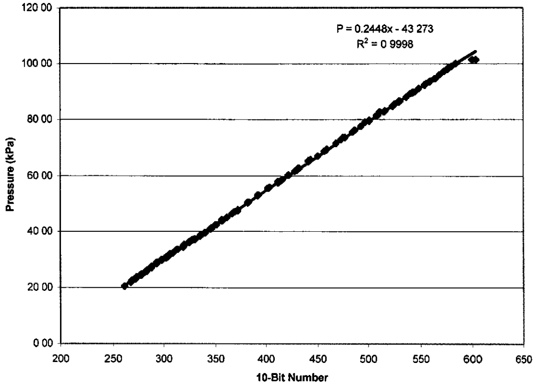


Figure 16. Chamber 1 Pressure Transducer Calibration

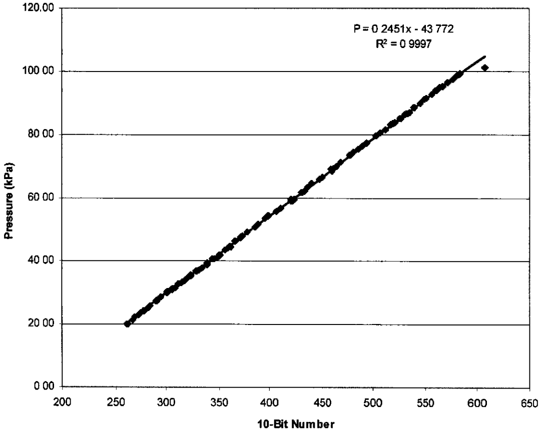


Figure 18. Chamber 3 Pressure Transducer Calibration

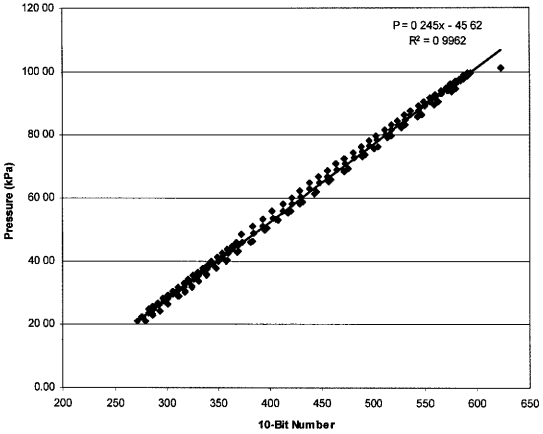


Figure 19. Chamber 4 Pressure Transducer Calibration

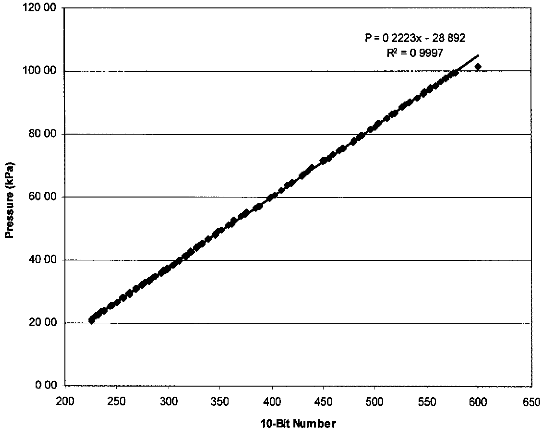


Figure 20. Chamber 5 Pressure Transducer Calibration

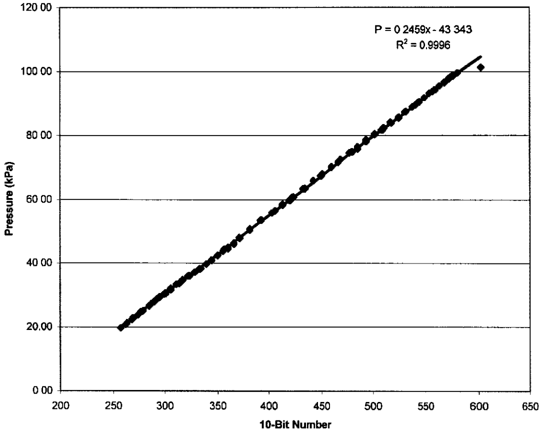


Figure 21. Chamber 6 Pressure Transducer Calibration

also to the amplifier circuit. As with the pressure calibration, the 10-bit number output by the A/D converter on the MCU was recorded and used to form the calibration equations. The temperature in the growth room was recorded using a pre-calibrated type T thermocouple temperature sensor. The 10-bit number was plotted versus actual room temperature to find the calibration equations. The C program for temperature calibration is in Appendix A. See Figures 22-27 for the temperature sensor calibration results, including the calibration equations.

MFCs

The three MFCs that are shared between each set of two chambers must be calibrated for the appropriate gas. These calibrations were performed by the manufacturer and the calibration gas was specified when the MFCs were ordered.

GCX

The GCX must also be calibrated periodically. The calibration gas contains with precisely measured amounts of the three component gases: nitrogen, oxygen, and carbon dioxide. The calibration gas and the air for the dummy cycle are on the same stream. The flow source is selected by a manual valve. The calibration procedure is initiated in the GCX-HMI software. Usually, the GCX must run through the calibration sequence two or more times before accurate readings are obtained.

System Checks

Several components of the system had to be checked as the system was being built. Serial communication between the PC and the MCUs was tested early in the

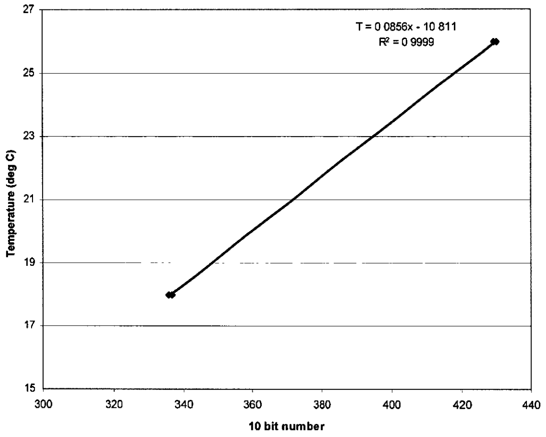


Figure 22. Chamber 1 Temperature Sensor Calibration

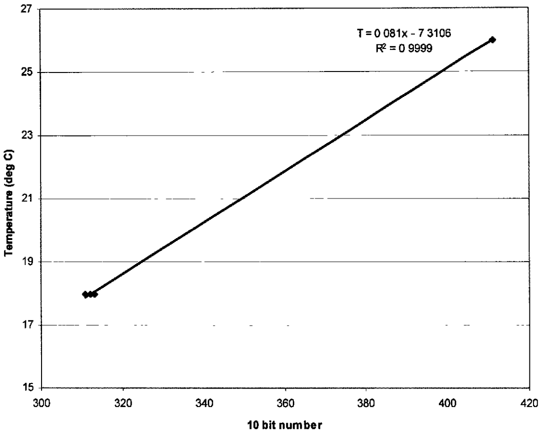


Figure 23. Chamber 2 Temperature Sensor Calibration

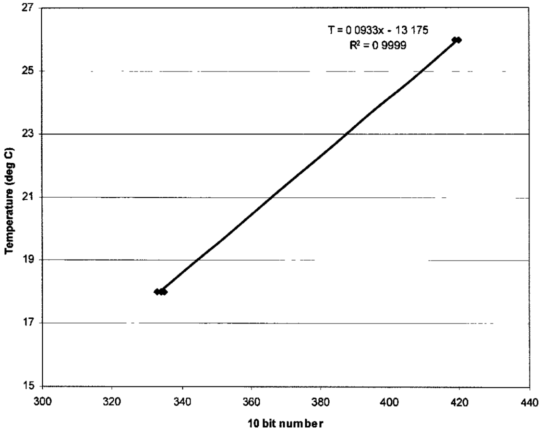


Figure 24. Chamber 3 Temperature Sensor Calibration

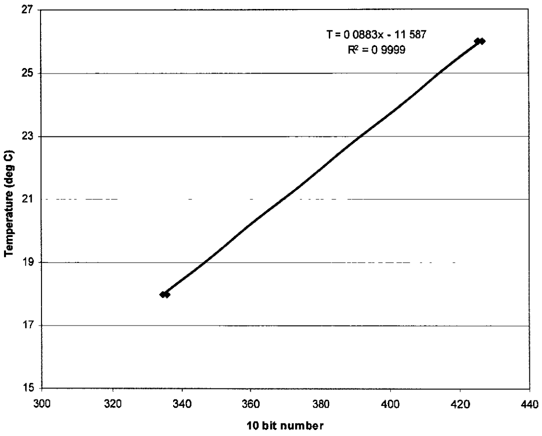


Figure 25. Chamber 4 Temperature Sensor Calibration

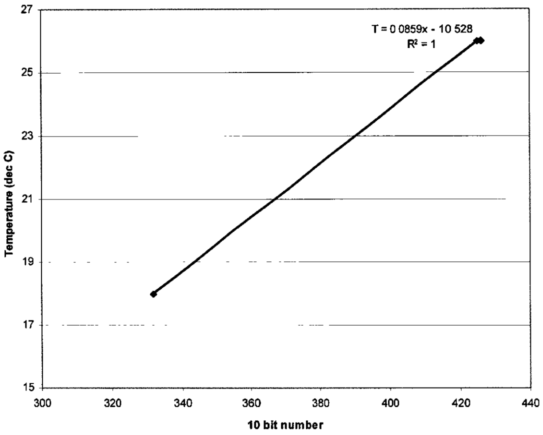


Figure 26. Chamber 5 Temperature Sensor Calibration

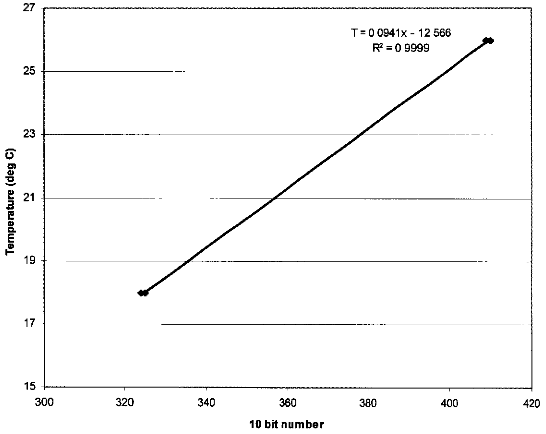


Figure 27. Chamber 6 Temperature Sensor Calibration

verified the correct flow rates were used and read by the MFCs because the amount of gas added is determined by a trapezoidal integration of the flowrate over time. If the amount of gas added is correct, then the flowrates that are the basis of the calculation must also be correct. (The other variables in the equation are time and several constants). A test similar to this was performed to check that the MCUs would add the correct amount of gas based on gas concentration conditions and setpoints.

The final test was to determine how accurate the addition of each gas to the chamber was. The amount of each gas added to the chamber was compared to the amount of gas that was supposed to be added. Both of these values were returned by the MCUs.

All of these tests were run on Chamber 2. Because same program and hardware are used for all chambers, only one chamber was necessary for testing. Program changes made as the result of these tests were implemented in the program on every MCU.

Modbus Communication

The final system check involved the GCX modbus. To first check communication between the modbus and PC, a short VI was written that performed a loopback test. This is function code 08. After communication was established, the GCX was turned on and allowed to run while several VIs read both the input and holding registers associated with each chamber. The gas concentrations returned to the PC were stored in files and later compared to concentrations saved in the GCX-HMI software.

System Shakedown

Once the system checks were used to verify that the various components of the system were working properly, a shakedown test was performed on chambers 3 and 4. The control programs for the two chambers were started up and allowed to run for 22 hours. During this test, the pressure and temperature data returned by the MCUs were recorded by the computer and plotted versus time to determine whether or not the system could maintain the pressure setpoint. For this test, the values for current gas concentrations and setpoint concentrations were programmed to be equal for the duration of the test. The pressure was set to 70 kPa for both chambers.

Results

The system checks both helped to determine the best way do accomplish the desired tests and later showed that the equipment worked properly. The most essential checks were those of the MCU program and of the modbus communication. The system shakedown tests were the final step in design of the LPPG system.

System Checks

MCU Program and MFCs

The calculations performed within the MCU program were checked. The microcontroller C compiler does not do math in the same way as typical compilers and many variables had to be forced to floating point type to get accurate values from the equations. With these corrections, it was found that the MCUs determined the correct amount of gas to add based on the pressure and gas concentrations currently in the chambers and the setpoints.

process by sending several characters to the MCU and then having it send those same characters back. Also, sensor outputs were checked using a voltmeter to determine if the output was reasonable before the calibrations were performed. The main system checks revolved around the MCU gas concentration calculations and the GCX.

MCU Program and MFCs

The MFCs and MCU program were checked simultaneously because the measured amount of each gas added to the chamber depends on the flow rate for that gas returned by its MFC. If the MFC reading was inaccurate for any gas, then the amount of gas the MCU believed it added (based on the flow rates of the gases) and amount of gas actually added (based in pressure in the chamber before and after gas addition) would not agree. But before this could be tested, other system checks were performed.

The first step was to check the MCU calculations for the amount of each gas to add. These calculations use the current conditions in the chamber and setpoints. The second step was to check the amount of gas actually added to each chamber versus the amount of gas that should have been added. To do this, a chamber was pulled down to a certain pressure, and then the pressure setpoint was raised. The MCU would then add gases in the appropriate concentrations to raise the pressure in the chamber. The pressures before and after addition of the gases were recorded. The MCU also returned the amount of each gas added. These amount of gas added based on pressure change was compared to the amount of gas added as reported by the MCU to check both the accuracy of the flow rates returned by the MFCs (which are set to certain values in the program) and to determine if the MCUs were adding the correct amount of gas. This test

The amount of gas added calculated by the MCU agreed with the amount of gas that must have been added to account for the pressure difference at that temperature. Table 4 shows for the results from this test. There are several tests where the difference between the MCU value and the PC value are very large. It was determined that this was because the pressure reading used for the MCU calculations was not the same pressure value returned by the MCUs.

Pressure is read several times in the program. The correct reading had to be used in the calculations for the amount of gas added based on pressure. This problem was corrected and additional tests were run. The error was still large in some instances, and the MCU was programmed to return the pressure value as a 10-bit number instead of in kilopascals. The computer then converted this number to kPa and the converted number was used in the CPU calculations. As a more precise pressure reading was used to check calculations, the error decreased until it was almost nonexistent. When the MCU returns a value for pressure in kilopascals or temperature in Celsius, this number has been rounded and is not as precise as the temperature measurement used in the MCU calculations. The spreadsheets used for the calculations are in Appendix C.
During the combined MFC and MCU program tests, the microcontroller returned 2 values for each gas: the amount of gas it was supposed to add, and the amount of gas actually added. There was always a difference between these numbers. The largest error was for carbon dioxide. This is understandable since carbon dioxide concentration is normally measured in parts per million, whereas oxygen and nitrogen concentration are measured as percentages. So though the CO_2 error looks large, it is in fact extremely

Table 4. Percent Difference Between the Amount of Gas Added Calculated by the MCU and the Amount of Gas Added Based on Pressure Difference. These numbers were compared to determine if the MCUs were adding the correct amount of each gas.

Pressures		CPU values	MCU values	% of Total Gas Added	% Difference
50	v_add[1]	1 0198	1.0675	0 1998	-4 67%
60	v_add[2]	0 0055	0 0053	0 0010	2 91%
	v_add[3]	4.4084	4 2702	0 7992	3.14%
60 1	v_add[1]	1 0910	1.0841	0.1998	0 63%
70	v_add[2]	0 0054	0 0054	0 0010	0.63%
	v_add[3]	4 3643	4 3369	0 7992	0.63%
70	v_add[1]	1 0198	1 0743	0 1998	-5.34%
80	v_add[2]	0 0055	0 0053	0 0010	2 52%
	v_add[3]	4.4084	4 2977	0 7992	2 51%
55 1	v_add[1]	0 5400	0.5382	0 1998	0 33%
60	v_add[2]	0 0027	0 0027	0 0010	0 37%
	v_add[3]	2 1601	2 1530	0 7992	0 33%
60 1	v_add[1]	0 3196	0 2601	0 1998	18 63%
63	v_add[2]	0 0016	0 0013	0 0010	18 65%
	v_add[3]	1 2784	1 0403	0 7992	18 63%
63 2	v_add[1]	0 3086	0 2995	0.1998	2 95%
66	v_add[2]	0 0015	0.0015	0.0010	2 94%
	v_add[3]	1 2344	1 1979	0.7992	2 95%
66 4	v_add[1]	0 3967	0 4226	0 1998	-6 51%
70	v_add[2]	0 0020	0.0021	0 0010	-6 50%
	v_add[3]	1 5870	1 6904	0 7992	-6 51%
70 2	v_add[1]	1 0799	1.0743	0 1998	0 52%
80	v_add[2]	0 0054	0 0053	0 0010	0 53%
	v_add[3]	4 3202	4 2977	0 7992	0.52%
70	v_add[1]	1 0799	0 9420	0 1998	12 77%
80	v_add[2]	0 0054	0 0047	0.0010	12 78%
	v_add[3]	4 3202	3 7684	0.7992	12 77%
71 4	v_add[1]	0 8894	0 8891	0 1998	0 03%
80	v_add[2]	0 0044	0.0044	0 0010	0 05%
	v_add[3]	3 5579	3 5567	0.7992	0 03%
71 45	v_add[1]	0 9423	0 9420	0.1998	0 03%
80	v_add[2]	0 0047	0 0047	0 0010	0 04%
	v_add[3]	3 7697	3.7684	0 7992	0 03%

small: a percentage of 1%. (The maximum measurable concentration of CO_2 is 1%.) The results are shown in Table 5, and because the control program is identical on all of the MCUs, it can be deduced that the results are similar for all of the chambers.

Modbus Communication

The initial modbus communication tests showed that not all of the components for all of the streams were being updated in the modbus. The stream 1 components and the nitrogen concentration for stream 5 were being updated in the modbus registers, but no other stream data was updated. This problem was solved by activating the modbus in the GCX-HMI software, which was overlooked during initial setup. The values read by LabVIEW from the modbus were then compared to the values saved by the GCX maintenance software. Tables 6, 7, and 8 show this data. Carbon dioxide showed the largest error between the GCX software concentration and the modbus concentration. The difference between the two numbers can be attributed to how data is stored in the modbus. The GCX maintenance software shows a concentration with a 4-decimal place precision for each component gas. However, the modbus only stores an integer between 0 and 4095 that represents a concentration. This integer is converted to a concentration in the Save GCX Data VI. The converted concentrations only agree with the GCX concentrations for the three significant digits. This was an acceptable degree of error in the gas concentrations.

System Shakedown

The pressure shakedown tests showed that the system could maintain pressure over a 22 hour period. Figure 28 shows the pressure vs. time graph for chambers 3 and

Table 5. Percent Difference Between the Amount of Gas to Add and the Amount of Gas Added. The MCUs return the amount of each gas that needs to be added to reach setpoint conditions, and the actual amount of gas added. These numbers were compared to determine if the MCUs were adding the correct amount of each gas.

O2			CO2			N2		
Liters to Add	Liters Added	Percent Difference	Liters to Add	Liters Added	Percent Difference	Liters to Add	Liters Added	Percent Difference
1.9413	1.9413	0.000013	0.0038	0.0038	0.002646	3.7017	3.7017	0.000007
2.7392	2.7392	0.000006	0.0046	0.0046	0.001959	4.2978	4.2977	0.000009
3.8009	3.8009	0.000001	0.0054	0.0053	0.002812	3.7684	3.7684	0.000005
0.9254	0.9253	0.000028	0.0047	0.0047	0.003634	3.5567	3.5567	0.000007
1.0743	1.0743	0.000022	0.0044	0.0044	0.003398	3.7684	3.7684	0.000002
0.9420	0.9420	0.000035	0.0047	0.0047	0.000428	3.9917	3.9917	0.000001
0.8891	0.8891	0.000038	0.0050	0.0050	0.002220	4.2702	4.2702	0.000005
0.9420	0.9420	0.000028	0.0053	0.0053	0.002452	4.3369	4.3369	0.000009
1.0675	1.0674	0.000030	0.0054	0.0054	0.003343	4.2977	4.2977	0.000003
1.0841	1.0841	0.000016	0.0053	0.0053	0.001874	2.1530	2.1530	0.000011
1.0743	1.0743	0.000021	0.0027	0.0027	0.002994	1.0403	1.0403	0.000021
0.5382	0.5382	0.000032	0.0013	0.0013	0.000000	1.1980	1.1979	0.000016
0.2601	0.2601	0.000131	0.0015	0.0015	0.012105	1.6904	1.6904	0.000014
0.2995	0.2995	0.000040	0.0021	0.0021	0.003337			
0.4226	0.4226	0.000083						
	Average error	0.000032			0.003086			0.000008

Table 6. Percent Difference Between GCX-HMI Software Oxygen Concentration and GCX Modbus Oxygen Concentration. The oxygen concentrations from each of the six chambers as shown in the GCX-HMI software are labeled as 1 to 6. The corresponding Modbus reading is shown in the following column.

1	modbus	Percent Difference	2	modbus	Percent Difference	3	modbus	Percent Difference	4	modbus	Percent Difference	5	modbus	Percent Difference	6	modbus	Percent Difference
21 919	21 921	0 01%	21.956	21 954	0 01%	21 949	21 947	0 01%	21 971	21.941	0 14%	21 993	21 953	0.18%	21 993	21 862	0 59%
21 832	21 833	0.01%	21 854	21.857	0 02%	21 949	21 855	0.43%	21 941	21 849	0 42%	21.956	21 909	0 21%	21 956	21 826	0 59%
21 854	21 855	0 01%	21.839	21.836	0 01%	21 854	21 814	0 18%	21 846	21 832	0 06%	21 912	21 807	0 48%	21.912	21 827	0 39%
21 861	21 860	0 00%	21.846	21.844	0 01%	21 817	21 829	0 06%	21 832	21 860	0 13%	21 810	21.837	0 13%	21 810	21 855	0 21%
21.839	21 837	0 01%	21.846	21 847	0 00%	21 832	21 865	0.15%	21 861	21 840	0 10%	21 839	21.827	0 05%	21 839	21 847	0 04%
21 780	21 778	0 01%	21 846	21 850	0.02%	21 868	21.785	0 38%	21 839	21 812	0 12%	21 824	21 771	0 24%	21 824	21 822	0 01%
21 802	21 803	0 00%	21 846	21 832	0.06%	21.788	21.806	0 08%	21 810	21 822	0 06%	21.773	21 759	0 06%	21 773	21.746	0 12%
21 766	21 764	0 01%	21 832	21 813	0.08%	21.810	21.757	0 24%	21.824	21 806	0 08%	21 758	21 795	0 17%	21 758	21 780	0 10%
21 766	21 804	0 18%	21 810	21 761	0 22%	21 758	21.767	0 04%	21 802	21 773	0.13%	21 795	21.749	0.21%	21 795	21 749	0 21%
21 802	21.736	0.30%	21 758	21 792	0.16%	21 766	21 741	0 11%	21.773	21 742	0.14%	21 751	21 784	0.15%	21.751	21 763	0.06%
Average		0 05%			0 06%			0 17%			0.14%			0 19%			0 23%

Table 7. Percent Difference Between GCX-HMI Software Nitrogen Concentration and GCX Modbus Nitrogen Concentration. The nitrogen concentrations from each of the six chambers as shown in the GCX-HMI software are labeled as 1 to 6. The corresponding Modbus reading is shown in the following column.

1	modbus	Percent Difference	2	modbus	Percent Difference	3	modbus	Percent Difference	4	modbus	Percent Difference	5	modbus	Percent Difference	6	modbus	Percent Difference
78 559	78 548	0 01%	78 657	78 651	0 01%	78 559	78 568	0 19%	78 633	78 517	0 15%	78 535	78 544	0 01%	77 998	78 493	0 64%
78 388	78.393	0 01%	78 462	78 473	0 01%	78 559	78 407	0 19%	78 510	78 481	0 04%	78 535	78 464	0 09%	78 486	78 463	0 03%
78 388	78 386	0 00%	78 437	78 430	0 01%	78 413	78 452	0 05%	78 486	78 414	0 09%	78 462	78 412	0 06%	78 462	78.353	0 14%
78 437	78 434	0 00%	78 388	78 388	0.00%	78.462	78 554	0 12%	78 413	78 552	0 18%	78 413	78.580	0 21%	78 364	78 529	0 21%
78 510	78 507	0 00%	78 388	78.590	0 26%	78 559	78 580	0 03%	78 559	78 479	0.10%	78 584	78 432	0 19%	78 535	78 456	0 10%
78 462	78 464	0 00%	78 584	78 580	0 00%	78 584	78 567	0 02%	78 486	78 561	0 10%	78 437	78 512	0 10%	78.462	78 455	0 01%
78 388	78 380	0 01%	78 584	78 484	0 13%	78 559	78 485	0 09%	78 559	78 472	0 11%	78 510	78 499	0 01%	78 462	78 467	0.01%
78 437	78 448	0 01%	78 486	78.512	0 03%	78 486	78 495	0 01%	78 462	78 481	0 02%	78.510	78 424	0 11%	78 462	78 414	0 06%
78 437	78 448	0 01%	78.510	78 543	0 04%	78 486	78 502	0.02%	78 486	78 494	0 01%	78 413	78 449	0 05%	78.413	78 426	0 02%
78 437	78 435	0 00%	78.535	78 497	0 05%	78.510	78.503	0 01%	78 486	78 384	0 13%	78 437	78.425	0.02%	78 437	78 425	0 02%
Average		0.01%			0 05%			0 07%			0.09%			0 09%			0.12%

Table 8. Percent Difference Between GCX-HMI Software Carbon Dioxide Concentration and GCX Modbus Carbon Dioxide Concentration. The carbon dioxide concentrations from each of the six chambers as shown in the GCX-HMI software are labeled as 1 to 6. The corresponding Modbus reading is shown in the following column.

1	modbus	Percent Difference	2	modbus	Percent Difference	3	modbus	Percent Difference	4	modbus	Percent Difference	5	modbus	Percent Difference	6	modbus	Percent Difference
0.037	0.037	1.07%	0.040	0.040	0.50%	0.042	0.042	0.00%	0.039	0.040	1.78%	0.037	0.038	2.43%	0.036	0.038	5.85%
0.038	0.038	0.78%	0.040	0.040	0.74%	0.042	0.042	0.00%	0.040	0.040	1.01%	0.038	0.038	0.26%	0.038	0.039	2.36%
0.038	0.038	0.26%	0.041	0.041	0.49%	0.042	0.043	2.87%	0.040	0.039	2.01%	0.038	0.039	2.36%	0.039	0.038	2.81%
0.039	0.039	1.04%	0.041	0.041	0.00%	0.043	0.042	1.18%	0.039	0.039	0.26%	0.039	0.038	2.06%	0.038	0.039	2.36%
0.038	0.038	0.26%	0.041	0.041	0.00%	0.042	0.043	1.90%	0.039	0.040	1.78%	0.038	0.039	1.83%	0.039	0.039	0.04%
0.038	0.038	0.26%	0.042	0.041	1.20%	0.043	0.044	3.53%	0.040	0.040	0.50%	0.039	0.039	1.04%	0.039	0.039	0.26%
0.039	0.039	1.04%	0.041	0.041	0.00%	0.044	0.043	1.60%	0.040	0.040	0.74%	0.039	0.040	3.09%	0.039	0.039	0.76%
0.039	0.039	0.76%	0.041	0.041	0.49%	0.043	0.044	3.04%	0.040	0.041	1.74%	0.040	0.039	1.52%	0.039	0.040	3.09%
0.039	0.040	1.78%	0.041	0.042	1.69%	0.044	0.044	0.45%	0.041	0.040	2.44%	0.039	0.040	3.09%	0.040	0.039	3.23%
0.040	0.039	2.50%	0.043	0.042	1.18%	0.044	0.044	0.69%	0.041	0.041	1.23%	0.040	0.040	0.50%	0.040	0.040	1.01%
Average		0.98%			0.63%			1.53%			1.35%			1.82%			2.18%

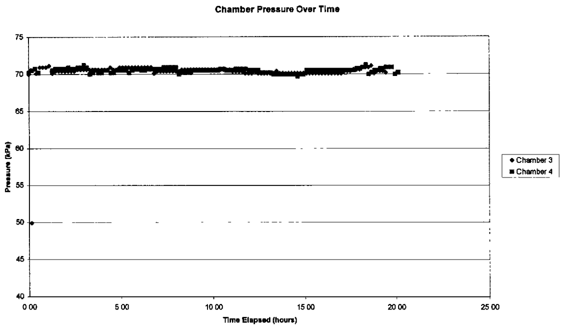


Figure 28. Pressure vs. Time for Shakedown Test Without Gas Addition

4. Chamber 3 started at 50 kPa and the system raised the pressure to 70 kPa. Chamber 4 was at 70 kPa at the start of the test. Pressure stayed close to the setpoint for the duration of the test. There are some variations in temperature in the chambers. These can be explained by looking at the graph of chamber temperatures for the duration of the test. Figure 29 shows the temperature in both chambers versus time.

The chamber temperature oscillates over a small range during the day. This is because the temperature control on the growth room is simple on/off control. Room temperature changes by small amounts as the air handling system comes on and shuts off. This has some slight effect on chamber pressure. In a fixed volume with a fixed amount of gas present, pressure decreases as temperature decreases and increases when temperature increases. However, if the temperature drops a significant amount, the chamber pressure will drop by a larger amount. During the night, chamber temperature drops and continues dropping until the lights in the chamber come back on.

Consequently, the pressure in the chambers drops over night as well. This is the small depression in the chamber pressure graph. If pressure changes too much, the MCUs will compensate by adding gases. The temperature and pressure graphs show that the system can correctly maintain pressure for several hours. It can compensate for temperature effects on pressure if the effects are great enough to move chamber pressure beyond the dead band around the setpoint.

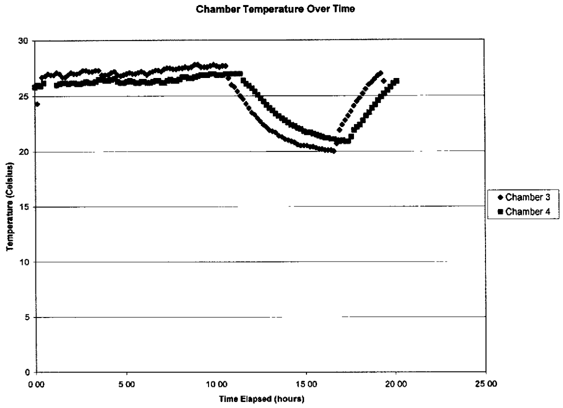


Figure 29. Temperature vs. Time for Shakedown Test Without Gas Addition

Discussion

System Checks

The system checks were a necessary step in start-up and development of the system. The tests exposed problems that needed to be corrected and allowed evaluation of various components of the entire LPPG system. The checks became part of the design process and verified proper operation of the system.

System Shakedown

The shakedown test shows that the system can maintain the pressure setpoint for several hours at a time. This is just a preliminary test. More tests at various pressures will be performed, and the gas concentration control portion of the system will be implemented in future tests. The gas concentration shakedown tests cannot yet be implemented due to problems maintaining serial communication between the PC and MCU when the amount of each gas added is returned by the MCU in addition to the pressure and temperature.

CONCLUSIONS AND FUTURE RESEARCH

Conclusions

The Generation II LPPG system is sufficient for running short-term tests as long as the atmosphere in the two chambers is recycled to prevent a build-up of ethylene and to replenish oxygen and carbon dioxide. The system can also be used to determine the effect of a non-replenished environment on plants. However, the lack of gas control limits the applications of this system.

The Generation III LPPG system has control over oxygen and carbon dioxide gas concentrations as well as pressure. Humidity and ethylene are removed from the chamber environment during the course of experiments. Long-term plant response tests can be conducted in the Generation III system. During these tests, pressure and gas concentrations can be adjusted, allowing implementation of various combinations of pressure and gas concentrations over the duration of the test. The system will be capable of dynamic adjustment of all relevant variables during experiments. However, some changes will have to be made before the full range of low pressure experiments can be run.

Future Research

Engineering

There are some changes that will be made to the Generation III system. The gas addition algorithm on the MCUs will be changed so that gases are added in small amounts in the correct proportions to prevent the pressure in the chambers from reaching levels much higher than atmospheric pressure. The current algorithm can result in large amounts of gas added to the chamber, sometimes raising pressure inside the chambers to two or three times atmospheric pressure. Because of the limited program space on the MCUs, this portion of the program will probably have to be moved to the PC and the LabVIEW chamber VIs will send the amount of each gas to add to the MCUs. The other solution is to find a new MCU with more program space and a new compiler with floating point math capability.

The prototype boards and circuits will be replaced by printed circuit boards. This will eliminate the possibility of a wire coming loose and impairing system operation. It will also provide the option of ordering extra boards so that damaged circuit boards can be removed and immediately replaced by a working board. This will decrease system down time should repairs need to be made.

Backup power will be provided to the MCUs, the GCX, and both computers to prevent them from being reset during the frequent power outages in the growth room. In this way, tests can continue to run even during power failure.

The serial communication algorithms will be fine tuned to prevent loss of communication between the PC and MCUs. At this point, when the gas addition portion of the MCU algorithm is used, the PC and MCU lose communication. This is because the MCU does not send the same number of characters every time. The number representing the volume of gas added will be formatted so that it always has the same number of characters. Zeroes will be used as place holders to the left of decimal. If this cannot be fit on the MCU with the existing program, then the calculation of how much of each gas to add will be moved to the LabVIEW chamber VIs and the volume to add will be sent with the pressure setpoint.

Tests will be conducted to model how plants affect carbon dioxide and oxygen concentrations in the chambers. If there is a large impact on concentrations over short periods of time, the GCX sampling frequency can be increased to sample each chamber every 15 minutes. Should this frequency still not be high enough to compensate for the amount of oxygen and carbon dioxide evolved by the plant, data from experiments with plants in the chamber will be used to create models of gas concentration values versus time. These models will then be incorporated into the LabVIEW chamber VI programs and setpoints will be adjusted to compensate for plant effects on gas concentrations over time. The MCU program execution frequency can be increased if necessary.

Plant Research

The system can maintain pressure and gas concentrations for plant experiments. The experiments run using the Generation III LPPG system will determine plant response to various combinations of pressure and oxygen and carbon dioxide concentrations. Models of ethylene production under various pressures and gas concentrations can also be developed. Both long term and short term tests with some of NASA's candidate crops, such as wheat and lettuce, will be conducted. This data will be used in the design a plant growth module for an extraplanetary habitat.

REFERENCES

1. Acevedo, M. F.; Waller, W.T. Modelling and control of a simple aquatic system. Ecological Modelling. 131:269-284; 2000.

2. Behrand, J.; Lane, H.W. Advanced life support program plan. NASA Document CTSD-ADV-348 (Revision B). Lyndon B. Johnson Space Center, Houston, TX; 1999.

3. Boston, P.J. Low-pressure greenhouses and plants for a manned research station on mars. Journal of the British Interplanetary Society. 34:189-192; 1981.

4. Chun, C.; Mitchell, C.A. Dynamic optimization of CELSS crop photosynthetic rate by computer-assisted feedback control. Adv. Space Res. 20(10):1855-1860; 1997.

5. Corey, K. A.; Barta, D. J.; Edeen, M. A.; Henninger, D. L. Atmospheric leakage and method for measurement of gas exchange rates of a crop stand at reduced pressure. Adv. Space Res. 20(10):1861-1867; 1997.

6. Corey, K. A.; Barta, D. J.; Henninger, D. L. Photosynthesis and respiration of a wheat stand at reduced atmospheric pressure and reduced oxygen. Adv. Space Res. 20(10):1869-1877; 1997.

7. Corey, K. A.; Bates, M. E.; Adams, S. L. Carbon dioxide exchange of lettuce plants under hypobaric conditions. Adv. Space Res. 18(4/5):265-272; 1996.

8. Daunicht, H. J.; Brinkjans, H. J. Plant response to reduced air pressure: advanced techniques and results. Adv. Space Res. 18(4/5):273-281; 1996.

9. Fleisher, D. H.; Baruh, H. An optimal control strategy for crop growth in advanced life support systems. Life Support Biosphere Sci. 8:43-53; 2001.

10. Fowler, P.A.; Wheeler, R. M.;Bucklin, R. A.; Corey, K.A. Low pressure greenhouse concepts for Mars. NASA Technical Memorandum 200-208577. Kennedy Space Center, FL; 2000; 116-127.

11. Goto, E. Environmental control for plant production in space CELSS. In: Goto, E.; Kurata, K.; Hayashi, M.; Sase, S., eds. Plant production in closed ecosystems. Netherlands: Kluwer Academic Publishers; 1997; 279-296.

12. Kacira, M.; Ling, P. P. Design and development of an automated and non-contact sensing system for continuous monitoring of plant health and growth. Trans. of ASAE. 44(4); 2001.

13. Langhans, R. W.; Tibbits, T. W., eds. Plant growth chamber handbook. North Central Region Research Publication No. 340. Iowa State University, Ames, IA; 1997.

14. Purswell, J. L. Engineering design of a hypobaric plant growth chamber. Master of Science Thesis: 89 pages. Biological and Agricultural Engineering. Texas A&M University, College Station, TX; 2002.

15. Saglio, P. H.; Rancillac, M.; Bruzan, F.; Pradet, A. Critical oxygen pressure for growth and respiration of excised and intact roots. Plant Physiology 76:151-154; 1984.

16. Salisbury, F. N. Challenges for bioregenerative life support of Mars. NASA Technical Memorandum 200-208577. Kennedy Space Center, FL; 2000; 18-26.

17. Schwartzkopf, S. H.; Grote, J. R.; Stroup, T.L. Design of a low atmospheric pressure plant growth chamber. SAE Technical Paper No. 951709. Warrendale, P.A.: Society of Automotive Engineers; 1995.

18. Schwartzkopf, S. H.; Mancinelli, R. L. Germination and growth of wheat in simulated Martian atmospheres. Acta Astronautica. 25(4):245-247; 1991.

19. Simpson, M. S.; Young, J. S. A plant growth structure for Martian derived atmosphere. SAE Technical Paper No. 981901. Warrendale, PA.: Society of Automotive Engineers; 1998.

20. Taiz; L.; Zeiger, E. Plant Physiology. Second Edition. Sunderland, MA. Sinauer Associates, Inc; 1998.

21. Weiland, P.O. Designing for human presence in space: an introduction to environmental control and life support systems. NASA Reference Publication 1324. Marshall Space Flight Center, Hunstville, AL; 1994.

22. Wheeler, R. M.; Mackowiak, C. L.; Stutte, G. W.; Sager, J. C.; Yorio, N. C.; Ruffe, L. M.; Fortson, R. E.; Dreschel, T. W.; Knott, W. M.; Corey, K. A., eds. Ethylene production by plants in a closed environment. Adv. Space Res. 18(4/5):193-196; 1996.

APPENDIX A: C CODE

Microcontroller Program

```
/*
        NASA low pressure plant growth chamber pressure
        and gas concentration control program for PIC16F877

        by. Andrew Hensley & Denise Brown
*/

#include <16F877.H>

#DEFINE CHAMBER_6  // chamber number to program the microchip for
#DEFINE V_Chamber 55 3          // volume of chamber in liters
#DEFINE GasConst 8.314// L-kPa/mol-K
#DEFINE rho_N2 1.25     // density in grams/liter
#DEFINE rho_O2 1.43     // density in grams/liter
#DEFINE rho_CO2 1.98    // density in grams/liter
#DEFINE MW_N2 28.01 // mollecular weight in grams/gram-mol
#DEFINE MW_O2 32.0  // mollecular weight in grams/gram-mol
#DEFINE MW_CO2 44.01            // mollecular weight in grams/gram-mol

#fuses  HS,NOWDT,NOPROTECT,PUT,BROWNOUT

#use delay(clock=10000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)

/* function prototypes */
void setup(void);
long getsetpt(void);
long getpressure(void);
long gettemperature(void);
long suck_air(long setpt);
long kPa2long(long p_kPa);
float long2kPa(long p_long);
float long2cel(long t_long);
float add_gas(int gas_number, float volume_to_add);
long check_way_over_p_set(long p_10bit, long p_set),
float calcmols(int gas_number, float n_gas, long conc);
//float find_max_mols2set(float p_kPa_set,float t_cel, float n_O2, float n_CO2, float n_N2, long O2_set,
long CO2_set, long N2_set),

main()
{
        /* declare variables */
        int i,pivot_gas;                        // int  = 8  bit unsigned
        long p_10bit,t_10bit,pmax,                      // long = 16 bit unsigned
        long p_kPa_set,O2_set,CO2_set,O2_conc,CO2_conc,N2_conc;                 // inputs
        long p_set,N2_set;                              // derived inputs
        float t_cel,p_kPa;                      // float = 32 bit floating point
        float n_gas,n_O2,n_N2,n_CO2,n_gas_set[4],n_add[4],          // float = 32 bit fp
        float v_add[4],volume_added;
        setup(); // set up timers, ADC, etc... see comments below


        /* begin main loop */
        while(1)
        {
                /* get setpoints from GC computer */
                OUTPUT_BIT(PIN_B7,1);           // debugging info -- high when waiting for setpoints
                p_kPa_set = getsetpt();    // get pressure setpoint in kPa
                O2_set   = getsetpt();     // get oxygen setpoint (parts per thousand)
                CO2_set  = getsetpt();     // get carbon dioxide setpoint (ppm)
                O2_conc  = getsetpt();     // get oxygen concentration from gc (parts per thousand)
                CO2_conc  = getsetpt();    // get carbon dioxide concentration from gc (ppm)
                N2_conc   = getsetpt(), // get nitrogen concentration from gc (parts per thousand)
                OUTPUT_BIT(PIN_B7,0);           // debugging info -- low when done reading setpoints

//              printf("%lu\t%lu\t%lu\t",p_kPa_set,O2_set,CO2_set);          // debug
//              printf("%lu\t%lu\t%lu\t",O2_conc,CO2_conc,N2_conc),          // debug

                /* convert to usable units */
                p_set=kPa2long(p_kPa_set);          //setpoint pressure in terms of a 10-bit number
                N2_set = 1000 - O2_set - CO2_set/1000,
                //calculate nitrogen setpoint (parts per thousand)

                /* read pressure and temperature */
                p_10bit=getpressure();
                t_10bit=gettemperature(),

                /* if pressure is >> than setpoint, adjust pressure */
                p_10bit=check_way_over_p_set(p_10bit,p_set);

                /* convert pressure and temperature for use in ideal gas eqn */
                p_kPa=long2kPa(p_10bit);             // convert pres to kPa float
                t_cel=long2cel(t_10bit);   // convert temp to celsius float

//              printf("%lu\t",p_10bit);   // debugging

                /* adjust concentrations [] */
                // gas #1 = Oxygen
                // gas #2 = Carbon Dioxide
                // gas #3 = Nitrogen

                // find total number of mols of gas in chamber by ideal gas law
                n_gas=p_kPa*V_Chamber/(GasConst*(t_cel+273 15));          // n=P*V/(R*T)

                // find number of mols of each component by concentration * total mols
                n_O2=calcmols(1,n_gas,O2_conc);
                n_CO2=calcmols(2,n_gas,CO2_conc),
                n_N2=calcmols(3,n_gas,N2_conc);

                // find the maximum number of mols needed by any gas to bring it to setpoint if that gas
                //is pivot gas
```

```
//pressure setpoint
n_gas_set[0]=(float)p_kPa_set*V_Chamber/(GasConst*(t_cel+273.15));

n_gas_set[1]=(float)n_O2/((float)O2_set/1000);      // oxygen
n_gas_set[2]=(float)n_CO2/(CO2_set/1000000 0);      // carbon dioxide
n_gas_set[3]=(float)n_N2/((float)N2_set/1000),      // nitrogen

pivot_gas=0;      // reset max
for(i=1,i<4,i++){
        if(n_gas_set[i]>n_gas_set[pivot_gas])
                pivot_gas=i;
}

// calculate how many mols of each gas to add
//new oxygen setpoint minus existing oxygen mols
n_add[1]=((float)O2_set/1000*n_gas_set[pivot_gas])-n_O2;
//new CO2 setpoint minus existing CO2 mols
n_add[2]=(CO2_set/1000000.0*n_gas_set[pivot_gas])-n_CO2;
//new nitrogen setpoint minus existing nitrogen mols
n_add[3]=((float)N2_set/1000*n_gas_set[pivot_gas])-n_N2;

// calculate how much volume that is at STP
v_add[1]=n_add[1]*MW_O2/rho_O2;                  // volume oxygen to add
v_add[2]=n_add[2]*MW_CO2/rho_CO2,     // volume carbon dioxide to add
v_add[3]=n_add[3]*MW_N2/rho_N2,                  // volume nitrogen to add

// add non-pivot gasses to chamber
volume_added=0;           //reset
OUTPUT_BIT(PIN_B6,1);          // debugging info -- high when adding gasses

for(i=1;i<4;i++)
{
        if(i!=pivot_gas)
        {       volume_added=add_gas(i,v_add[i]);
                printf("%f\t",volume_added);
        }
        else
                printf("0.000000\t");
}
OUTPUT_BIT(PIN_B6,0);          // debugging info -- low when not adding gasses

/* adjust pressure */
pmax=1.025*p_kPa_set;  //temporarily storing dead band above set point [kPa] in pmax
//convert pmax [kPa] to pmax [long] for comparison to p_10bit
pmax=kPa2long(pmax);

p_10bit=getpressure();     //check pressure now that gasses have been added

//              printf("%lu\t",p_10bit);    // debugging

if(p_10bit>pmax)//if pressure is above setpoint + margin, draw down to setpoint
        p_10bit=suck_air(p_set);
```

```
			/* read pressure and temperature again */
			p_10bit=getpressure();
			// this shouldn't be necessary because pressure was measured in suck_air()

			t_10bit=gettemperature();

			OUTPUT_BIT(PIN_C5,1),
			// debugging info -- high when about to write data, stays high if program reaches this
			// point

			/* send data to computer */
			p_kPa=long2kPa(p_10bit);
			t_cel=long2cel(t_10bit);

			//put tabs between outputs
			printf("%6 1f\t%6.1f\t",p_kPa,t_cel);
		}
}

void setup()
{
		/* configure analog inputs */
		SETUP_ADC( ADC_CLOCK_INTERNAL );
		SETUP_ADC_PORTS(ALL_ANALOG);
		// RA0 RA1 RA2 RA3 RA5 RE0 RE1 RE2 Ref=Vdd

		//use this command before using READ_ADC() to read channel 0 (pressure):
		set_adc_channel( 0 );

		// Channel 0, RA0, Pin  2, Chamber 1 Pressure
		// Channel 1, RA1, Pin  3, Chamber 1 Temperature
		// Channel 2, RA2, Pin  4, Oxygen Flow Controller
		// Channel 3, RA3, Pin  5, Carbon Dioxide Flow Controller
		// Channel 4, RA5, Pin  7, Nitrogen Flow Controller

		/* configure digital I/O */
		// RD3, Pin 19, vacuum pump on/off switch  on=high		output
		// RD1, Pin 20, vacuum pump valve		open=high		output
		// RD2, Pin 21, gas in valve				open=high		output
		// RB1, Pin 34, valve-open signal pin			open=high		output
		// RB2, Pin 35, valve-open check pin			open=high		input
		// RB4, Pin 37, CO2 flow controller (PWM eqiv.)	open=high		output

		/* old demultiplexing idea -- not used right now */
		// RB4, Pin 37, PWM demultiplexer -> O2		on=high		output
		// RB5, Pin 38, PWM demultiplexer -> CO2 on=high		output

		/* configure PWM */
		SETUP_CCP1(CCP_PWM);				// setup comparator 1 to PWM mode
		SETUP_CCP2(CCP_PWM);				// setup comparator 2 to PWM mode
		SETUP_TIMER_2(T2_DIV_BY_1,255,1);  // PWM uses timer 2
		// RC2, Pin 17, PWM1 -- demultiplexed oxygen and carbon dioxide flow controllers
		// RC1, Pin 16, PWM2 -- nitrogen flow controller
```

```
        //use command. SET_PWM1_DUTY(value) for PWM1 (10-bit value max) (an 8-bit value will be
        //left justified)
        //use command: SET_PWM2_DUTY(value) for PWM2 (10-bit value max) (an 8-bit value will be
        //left justified)
        /* initialize both PWM channels to 0 */
        SET_PWM1_DUTY(0);
        SET_PWM1_DUTY(0);
}


long getsetpt()
{
        char thous,hunds,tens,ones;
        long setpt=0;

        /* this subroutine expects to receive a 4 digit number converted to a 4 char string
        with leading spaces if the number requires fewer digits to represent it  */

        thous=getc();      //this will wait for the RS232 input indefinitely
        hunds=getc();
        tens =getc();
        ones =getc(),
        if(thous>47)
                setpt=1000 0*(thous-48);  //48 is the ascii value of '0'
        if(hunds>47)
                setpt+=100 0*(hunds-48);
        if(tens>47)
                setpt+=10 0*(tens-48);
        if(ones>47)
                setpt+=ones-48,
        return setpt;
}

long suck_air(setpt)
{
        int i;
        long p=1024,p_sum=0;
        OUTPUT_BIT(PIN_D3,1);           //vac pump on
        DELAY_MS(500),                  //delay .5 sec to suck air out of lines
        OUTPUT_BIT(PIN_D1,1);           //open vacuum pump valve
        SET_ADC_CHANNEL(0);             //ADC set to read pressure
        DELAY_US(1);            //wait for ADC to be ready
        while(p>setpt)
        {
                p_sum=0;
                for(i=0;i<20;i++)
                        p_sum+=READ_ADC();  //averaging out pressure transducer A/D
                p=p_sum/20;
        }
        OUTPUT_BIT(PIN_D3,0);           //vac pump off
        OUTPUT_BIT(PIN_D1,0);           //close chamber out valve
        return p;
}
```

```
long kPa2long(p_kPa)
{
        long p;
        //insert formula here to convert from kPa to 10-bit value representing calibrated pressure sensor's
        //output as will be read by the A/D converter taking into account the reference voltages supplied
        //pressure in kPa = 48.292*V - 48.173 where V=voltage.  we're using a 16-bit (long) variable to
        //hold the value from a 10-bit A/D converter on chip.  So for a voltage from 0 to 5 V, we will get
        //an  integer from 0 to 1023 from the A/D converter (because we'll be using 0 & 5 V as
        //reference)
        //re-arranging the above equation and including the integer to voltage conversion (5/1023) we get:

#IFDEF CHAMBER_1
        p=(p_kpa+43.273)/.2448;           //chamber #1
#ENDIF
#IFDEF CHAMBER_2
        p=(p_kPa + 44.962)/.2491;         //chamber #2
#ENDIF
#IFDEF CHAMBER_3
        p=(p_kPa + 43.772)/.2451;         //chamber #3
#ENDIF
#IFDEF CHAMBER_4
        p=(p_kPa + 45.62)/.245;           //chamber #4
#ENDIF
#IFDEF CHAMBER_5
        p=(p_kPa + 28.892)/.2223;         //chamber #5
#ENDIF
#IFDEF CHAMBER_6
        p=(p_kPa + 43.343)/.2459;         //chamber #6
#ENDIF
        return p;
}

float long2kPa(p_long)
{
        float p_kPa,
        //insert formula here to convert from 10-bit value to kPa.  10-bit value represents calibrated
        //pressure sensor's output as will be read by the A/D converter taking into account the reference
        //voltages supplied

#IFDEF CHAMBER_1
        p_kPa=0.2448*p_long - 43.273;     //chamber #1
#ENDIF
#IFDEF CHAMBER_2
        p_kPa=0.2491*p_long - 44.962;     //chamber #2
#ENDIF
#IFDEF CHAMBER_3
        p_kPa=0.2451*p_long - 43.772,     //chamber #3
#ENDIF
#IFDEF CHAMBER_4
        p_kPa=0.245*p_long - 45.62;       //chamber #4
#ENDIF
#IFDEF CHAMBER_5
```

```
        p_kPa=0.2223*p_long - 28.892;     //chamber #5
#ENDIF
#IFDEF CHAMBER_6
        p_kPa=0.2459*p_long - 43.343;     //chamber #6
#ENDIF
        return p_kPa;
}

float long2cel(t_long)
{
        float t_cel=0.0;
        //insert formula here to convert from 10-bit value to celsius. 10-bit value represents calibrated
        //thermistor's output as will be read by the A/D converter taking into account the reference
        //voltages supplied

#IFDEF CHAMBER_1
        t_cel=.0856*t_long - 10.811;        //chamber #1
#ENDIF
#IFDEF CHAMBER_2
        t_cel=.081*t_long - 7 3106,         //chamber #2
#ENDIF
#IFDEF CHAMBER_3
        t_cel=.0933*t_long - 13.175;        //chamber #3
#ENDIF
#IFDEF CHAMBER_4
        t_cel=.0883*t_long - 11.587;        //chamber #4
#ENDIF
#IFDEF CHAMBER_5
        t_cel=.0859*t_long - 10.528;        //chamber #5
#ENDIF
#IFDEF CHAMBER_6
        t_cel=.0941*t_long - 12.566;        //chamber #6
#ENDIF


        return t_cel,

}

float add_gas(gas_number, volume_to_add)
{
        /* This subroutine should check which gas is to be added, and how much to add, then
         * open the appropriate valves for the appropriate amount of time (with feedback) to
         * do so.
         * gas #1 = Nitrogen
         * gas #2 = Carbon Dioxide
         * gas #3 = Oxygen
         */
        /* declare variables */
        int q=0,q_prev=0,dt=0,mfc_channel,valve_pin,PWM1duty,PWM2duty,flowscale=1;
        long loopcount=0;
        float volume_added=0,
	if(volume_to_add<0.0000001)	//if no gas is supposed to be added, skip rest of subroutine
		return 0.0;

	/* wait for other MCU to finish using its gas inlet valve */
	while(INPUT(PIN_B2)),
	OUTPUT_BIT(PIN_B1,1);		//declare gas inlet valves in use by you

	/* set which pins to use based on gas # */
	if(gas_number == 1){		//Oxygen
		mfc_channel= 4;		//the channel that the A/D converter will read to find flowrate
		PWM1duty=0;		// not using PWM2
		PWM2duty=255;		// open MFC all the way = 1000 ml/min
	}
	if(gas_number == 2){		//Carbon Dioxide
		mfc_channel= 3;		//the channel that the A/D converter will read to find flowrate
		OUTPUT_BIT(PIN_B4,1);
		//open MFC all the way = 500 ml/min  !!!carbon dioxide flow controller differenet flow
		//rate
		PWM1duty=0;		//not using PWM1
		PWM2duty=0;		//not using PWM2
		flowscale=2;		//account for flowrate being half of normal by dividing by
					//flowscale
	}
	if(gas_number == 3){		//Nitrogen
		mfc_channel= 2;		//the channel that the A/D converter will read to find flowrate
		PWM1duty=255;		// open MFC all the way = 1000 ml/min
		PWM2duty=0;		// not using PWM2
	}

	SETUP_COUNTERS(RTCC_INTERNAL,RTCC_DIV_32);
	//there are problems if this counter resets, messes up dt

	SET_ADC_CHANNEL(mfc_channel);	//ready to read MFC
	OUTPUT_BIT(PIN_D2,1);		//open gas in valve
	DELAY_MS(10),			//wait 10 milliseconds for any air in the line to drain
					//to chamber
	SET_PWM1_DUTY(PWM1duty); //begin to let appropriate flow of gas through O2 MFC
	SET_PWM2_DUTY(PWM2duty); //begin to let appropriate flow of gas through N2 MFC
	SET_RTCC(0);		//reset clock to zero

	/* trapezoidal integration to control volume_added */
	while(volume_added < volume_to_add)
	{
		q_prev=q;		//saving prev q for trap integration
		dt=GET_RTCC();			//change in time = clock value b/c we reset each loop
		SET_RTCC(0);		//reset clock to zero
		q=READ_ADC(),			//read MFC -- flow rate

/*		constant below (8.3660131E-10) arrived at by:

RTCC cnts*A/D flowrate | 32 instr cycles |  4 clk cycles  |     1 s          | 1 min  | 1 Liter/minute
-----------------------------------------------------------------------------------------------------------
                       |1 RTCC count    | instr cycle     | 10^7 clk cycles  | 60 s   | 255 A/D value
*/
        volume_added+=((8 3660131E-10)*((float)q+(float)q_prev)/2.0*(float)dt)/(float)flowscale,
        //volume = flow rate * time (trapezoidal integration)
                loopcount++;
        }

        /* close out valves, etc.. return to base state */
        SET_PWM1_DUTY(0),               //close O2 MFC internal valve
        SET_PWM2_DUTY(0);               //close N2 MFC internal valve
        OUTPUT_BIT(PIN_B4,0);                   //close CO2 MFC if it was open
        DELAY_MS(10);                           //wait 10 milliseconds for any air in the line to drain
                                                //to chamber
        OUTPUT_BIT(PIN_D2,0),                   //close gas in valve
        OUTPUT_BIT(PIN_B1,0);                   //declare gas inlet valves no longer in use by you

//      printf("%u\t%u\t%u\t%lu\t",q,q_prev,dt,loopcount); //debugging
//      printf("%f liters of gas number %u added, %f liters requested\t", volume_added, gas_number,
                volume_to_add); //debugging notes

        return volume_added,
}

long getpressure()
{
/* This subroutine reads the appropriate A/D channel and returns pressure as a 10-bit number (long) */
        int i;
        long p_10bit,p_sum=0;

        SET_ADC_CHANNEL(0),
        DELAY_US(1),
        for(i=0,i<50;i++)
                p_sum+=READ_ADC(); //averaging out pressure transducer A/D to be more accurate

        p_10bit=p_sum/50,

        return p_10bit;
}

long gettemperature()
{
/* This subroutine reads the appropriate A/D channel and returns temperature as a 10-bit number (long) */
        int i,
        long t_10bit,t_sum=0,

        SET_ADC_CHANNEL(1);
        DELAY_US(1);
        for(i=0;i<10;i++)
                t_sum+=READ_ADC();  //averaging out thermistor A/D to be more accurate
        t_10bit=t_sum/10,

        return t_10bit;
}
```

```
long check_way_over_p_set(p_10bit, p_set)
{
	/* This subroutine checks if pressure is >> than setpoint, then adjusts pressure if it is */
		if((p_10bit>p_set)&&((p_10bit-p_set)>150))
			p_10bit=suck_air(p_set),

	return p_10bit,
}

float calcmols(gas_number,n_gas,conc)
{
	float n_mols;
	n_mols=((float)conc/1000)*n_gas,
	if(gas_number==2)
		n_mols/=1000;	// carbon dioxide concentration was in ppm not parts per thousand

	return n_mols,
}
```

Calibration Programs

Decreasing Pressure

```
#include <16F877 H>

#fuses  HS,NOWDT,NOPROTECT,PUT,BROWNOUT

#use delay(clock=10000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)

/* function prototypes */
long getpressure(void);
long gettemperature(void);

main()
{
	/* declare variables */
	int i,;					// int  = 8  bit unsigned
	long p_10bit,t_10bit,;			// long = 16 bit unsigned

	setup(); // set up timers, ADC, etc... see comments below
	p_10bit=getpressure();	//read atmospheric pressure
	printf("%lu\n",p_10bit)	//output to computer


	while(p_10bit>200)	//continue loop until pressure drops below 30kPa
				//the value will vary for each chamber
				//the 10-bit number used is based on previous tests
	{
		OUTPUT_HIGH(PIN_D1),		//open vacuum valve
		DELAY_MS(100);			//pull air out of lines
```

```
            OUTPUT_HIGH(PIN_D0);        //turn on pump

            DELAY_MS(2000);             //leave pump on for 2 seconds

            OUTPUT_LOW(PIN_D1);         //close valve
            OUTPUT_LOW(PIN_D0);         //turn off pump

            DELAY_MS(1000);             //wait for pressure transducer to settle
                                        //pressure transducer has a second order response

            p_10bit=getpressure();      //read pressure
            printf("%lu\n",p_10bit)     //send to computer
        }
}

void setup()
{
        /* configure analog inputs */
        SETUP_ADC( ADC_CLOCK_INTERNAL );
        SETUP_ADC_PORTS(ALL_ANALOG),
        // RA0 RA1 RA2 RA3 RA5 RE0 RE1 RE2 Ref=Vdd

        //use this command before using READ_ADC() to read channel 0 (pressure)·
        set_adc_channel( 0 );

        // Channel 0, RA0, Pin  2, Chamber 1 Pressure
        // Channel 1, RA1, Pin  3, Chamber 1 Temperature
        // Channel 2, RA2, Pin  4, Oxygen Flow Controller
        // Channel 3, RA3, Pin  5, Carbon Dioxide Flow Controller
        // Channel 4, RA5, Pin  7, Nitrogen Flow Controller

        /* configure digital I/O */
        // RD0, Pin 19, vacuum pump on/off switch  on=high         output
        // RD1, Pin 20, vacuum pump valve          open=high       output
        // RD2, Pin 21, gas in valve                       open=high       output
        // RB1, Pin 34, valve-open signal pin              open=high       output
        // RB2, Pin 35, valve-open check pin               open=high       input
        // RB4, Pin 37, CO2 flow controller (PWM eqiv.)    open=high       output

        /* old demultiplexing idea -- not used right now */
        // RB4, Pin 37, PWM demultiplexer -> O2            on=high         output
        // RB5, Pin 38, PWM demultiplexer -> CO2 on=high           output

        /* configure PWM */
        SETUP_CCP1(CCP_PWM);                    // setup comparator 1 to PWM mode
        SETUP_CCP2(CCP_PWM),                    // setup comparator 2 to PWM mode
        SETUP_TIMER_2(T2_DIV_BY_1,255,1); // PWM uses timer 2
        // RC2, Pin 17, PWM1 -- demultiplexed oxygen and carbon dioxide flow controllers
        // RC1, Pin 16, PWM2 -- nitrogen flow controller
        //use command: SET_PWM1_DUTY(value) for PWM1 (10-bit value max) (an 8-bit value will be
        //left justified)
        //use command: SET_PWM2_DUTY(value) for PWM2 (10-bit value max) (an 8-bit value will be
        //left justified)
        /* initialize both PWM channels to 0 */
        SET_PWM1_DUTY(0);
        SET_PWM1_DUTY(0);
}

long getpressure()
{
/* This subroutine reads the appropriate A/D channel and returns pressure as a 10-bit number (long) */
        int i;
        long p_10bit,p_sum=0,

        SET_ADC_CHANNEL(0);
        DELAY_US(1),
        for(i=0;i<50;i++)
                p_sum+=READ_ADC();  //averaging out pressure transducer A/D to be more accurate

        p_10bit=p_sum/50,

        return p_10bit;
}

long gettemperature()
{
/* This subroutine reads the appropriate A/D channel and returns temperature as a 10-bit number (long) */
        int i,
        long t_10bit,t_sum=0,

        SET_ADC_CHANNEL(1);
        DELAY_US(1);
        for(i=0;i<10;i++)
                t_sum+=READ_ADC(),  //averaging out thermistor A/D to be more accurate
        t_10bit=t_sum/10;

        return t_10bit,
}
```

Increasing Pressure

```
#include <16F877.H>

#fuses  HS,NOWDT,NOPROTECT,PUT,BROWNOUT

#use delay(clock=10000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)

/* function prototypes */
long getpressure(void),
long gettemperature(void);
long suck_air(long setpt);
```

```
main()
{
        /* declare variables */
        int i,;                                         // int  = 8  bit unsigned
        long p_10bit,t_10bit,;                          // long = 16 bit unsigned

        setup(),  // set up timers, ADC, etc... see comments below

        suck_air(200)            //pull pressure down to 30 kPa

        p_10bit=getpressure(),   //read lowest pressure
        printf("%lu\n",p_10bit)  //output to computer


        while(p_10bit<400)       //continue loop until pressure reaches atmospheric

        {
                OUTPUT_HIGH(PIN_D2),        //open gas in valve – which lets air out of chamber

                DELAY_MS(2000),             //let air enter chamber for 2 seconds

                OUTPUT_LOW(PIN_D2);         //close valve

                DELAY_MS(1000);             //wait for pressure transducer to settle
                                            //pressure transducer has a second order response

                p_10bit=getpressure();      //read pressure
                printf("%lu\n",p_10bit)     //send to computer
        }
}

void setup()
{
        /* configure analog inputs */
        SETUP_ADC( ADC_CLOCK_INTERNAL );
        SETUP_ADC_PORTS(ALL_ANALOG);
        // RA0 RA1 RA2 RA3 RA5 RE0 RE1 RE2 Ref=Vdd

        //use this command before using READ_ADC() to read channel 0 (pressure):
        set_adc_channel( 0 );

        // Channel 0, RA0, Pin  2, Chamber 1 Pressure
        // Channel 1, RA1, Pin  3, Chamber 1 Temperature
        // Channel 2, RA2, Pin  4, Oxygen Flow Controller
        // Channel 3, RA3, Pin  5, Carbon Dioxide Flow Controller
        // Channel 4, RA5, Pin  7, Nitrogen Flow Controller

        /* configure digital I/O */
        // RD0, Pin 19, vacuum pump on/off switch  on=high        output
        // RD1, Pin 20, vacuum pump valve          open=high      output
        // RD2, Pin 21, gas in valve                       open=high      output
        // RB1, Pin 34, valve-open signal pin              open=high      output
        // RB2, Pin 35, valve-open check pin               open=high      input
        // RB4, Pin 37, CO2 flow controller (PWM eqiv )      open=high          output

        /* old demultiplexing idea -- not used right now */
        // RB4, Pin 37, PWM demultiplexer -> O2              on=high            output
        // RB5, Pin 38, PWM demultiplexer -> CO2 on=high               output

        /* configure PWM */
        SETUP_CCP1(CCP_PWM);                          // setup comparator 1 to PWM mode
        SETUP_CCP2(CCP_PWM);                          // setup comparator 2 to PWM mode
        SETUP_TIMER_2(T2_DIV_BY_1,255,1); // PWM uses timer 2
        // RC2, Pin 17, PWM1 -- demultiplexed oxygen and carbon dioxide flow controllers
        // RC1, Pin 16, PWM2 -- nitrogen flow controller
        //use command: SET_PWM1_DUTY(value) for PWM1 (10-bit value max) (an 8-bit value will be
        //left justified)
        //use command: SET_PWM2_DUTY(value) for PWM2 (10-bit value max) (an 8-bit value will be
        //left justified)
        /* initialize both PWM channels to 0 */
        SET_PWM1_DUTY(0),
        SET_PWM1_DUTY(0),
}

long suck_air(setpt)
{
        int i;
        long p=1024,p_sum=0;
        OUTPUT_BIT(PIN_D3,1);          //vac pump on
        DELAY_MS(500);                 //delay 5 sec to suck air out of lines
        OUTPUT_BIT(PIN_D1,1);          //open vacuum pump valve
        SET_ADC_CHANNEL(0);            //ADC set to read pressure
        DELAY_US(1);           //wait for ADC to be ready
        while(p>setpt)
        {
                p_sum=0;
                for(i=0;i<20;i++)
                        p_sum+=READ_ADC();  //averaging out pressure transducer A/D
                p=p_sum/20;
        }
        OUTPUT_BIT(PIN_D3,0);          //vac pump off
        OUTPUT_BIT(PIN_D1,0);          //close chamber out valve
        return p;
}


long getpressure()
{
/* This subroutine reads the appropriate A/D channel and returns pressure as a 10-bit number (long) */
        int i;
        long p_10bit,p_sum=0;

        SET_ADC_CHANNEL(0),
        DELAY_US(1);
        for(i=0;i<50;i++)
                p_sum+=READ_ADC();  //averaging out pressure transducer A/D to be more accurate
        p_10bit=p_sum/50;

        return p_10bit;
}

long gettemperature()
{
/* This subroutine reads the appropriate A/D channel and returns temperature as a 10-bit number (long) */
        int i,
        long t_10bit,t_sum=0;

        SET_ADC_CHANNEL(1);
        DELAY_US(1);
        for(i=0;i<10;i++)
                t_sum+=READ_ADC();   //averaging out thermistor A/D to be more accurate
        t_10bit=t_sum/10;

        return t_10bit;
}
```

Temperature

```
#include <16F877.H>

#fuses  HS,NOWDT,NOPROTECT,PUT,BROWNOUT

#use delay(clock=10000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)

/* function prototypes */
long gettemperature(void);

main()
{
        /* declare variables */
        int i,;                                 // int  = 8  bit unsigned
        long p_10bit,t_10bit,,                  // long = 16 bit unsigned

        setup(); // set up timers, ADC, etc... see comments below

        //the chip will be reset whenever temperature changes and will output 10 ten-bit numbers

        for(i=0;i<10,i++)
        {
                t_10bit=gettemperature();       //read temperature
                printf("%lu\n",t_10it),         //output to computer
        }
}
```

```
void setup()
{
	/* configure analog inputs */
	SETUP_ADC( ADC_CLOCK_INTERNAL );
	SETUP_ADC_PORTS(ALL_ANALOG),
	// RA0 RA1 RA2 RA3 RA5 RE0 RE1 RE2 Ref=Vdd

	//use this command before using READ_ADC() to read channel 0 (pressure):
	set_adc_channel( 0 );

	// Channel 0, RA0, Pin  2, Chamber 1 Pressure
	// Channel 1, RA1, Pin  3, Chamber 1 Temperature
	// Channel 2, RA2, Pin  4, Oxygen Flow Controller
	// Channel 3, RA3, Pin  5, Carbon Dioxide Flow Controller
	// Channel 4, RA5, Pin  7, Nitrogen Flow Controller

	/* configure digital I/O */
	// RD0, Pin 19, vacuum pump on/off switch  on=high          output
	// RD1, Pin 20, vacuum pump valve          open=high        output
	// RD2, Pin 21, gas in valve                         open=high        output
	// RB1, Pin 34, valve-open signal pin                open=high        output
	// RB2, Pin 35, valve-open check pin                 open=high        input
	// RB4, Pin 37, CO2 flow controller (PWM eqiv.)      open=high        output

	/* old demultiplexing idea -- not used right now */
	// RB4, Pin 37, PWM demultiplexer -> O2              on=high          output
	// RB5, Pin 38, PWM demultiplexer -> CO2 on=high            output

	/* configure PWM */
	SETUP_CCP1(CCP_PWM);                          // setup comparator 1 to PWM mode
	SETUP_CCP2(CCP_PWM);                          // setup comparator 2 to PWM mode
	SETUP_TIMER_2(T2_DIV_BY_1,255,1); // PWM uses timer 2
	// RC2, Pin 17, PWM1 -- demultiplexed oxygen and carbon dioxide flow controllers
	// RC1, Pin 16, PWM2 -- nitrogen flow controller
	//use command: SET_PWM1_DUTY(value) for PWM1 (10-bit value max) (an 8-bit value will be
	//left justified)
	//use command: SET_PWM2_DUTY(value) for PWM2 (10-bit value max) (an 8-bit value will be
	//left justified)
	/* initialize both PWM channels to 0 */
	SET_PWM1_DUTY(0);
	SET_PWM1_DUTY(0);
}


long gettemperature()
{
/* This subroutine reads the appropriate A/D channel and returns temperature as a 10-bit number (long) */
	int i,
	long t_10bit,t_sum=0;

	SET_ADC_CHANNEL(1);
	DELAY_US(1),
	for(i=0,i<10,ı++)
		t_sum+=READ_ADC(),	//averaging out thermistor A/D to be more accurate
	t_10bit=t_sum/10,

	return t_10bit;
}
```

APPENDIX B: GENERATION III LPPG SYSTEM WIRING DIAGRAMS

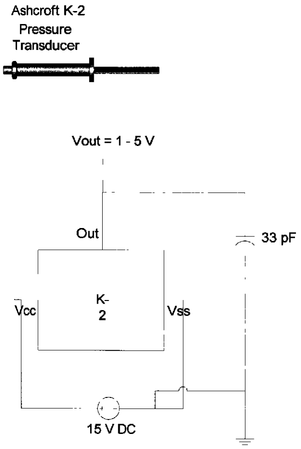


Figure B-2. Pressure transducer wiring. V_{out} goes into the MCU.

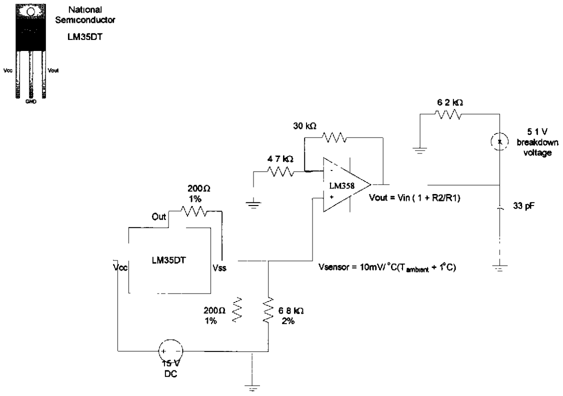


Figure B-3. Temperature sensor and signal conditioning circuit. V_{out} goes to the MCU.

MCU and Serial Wiring Diagram

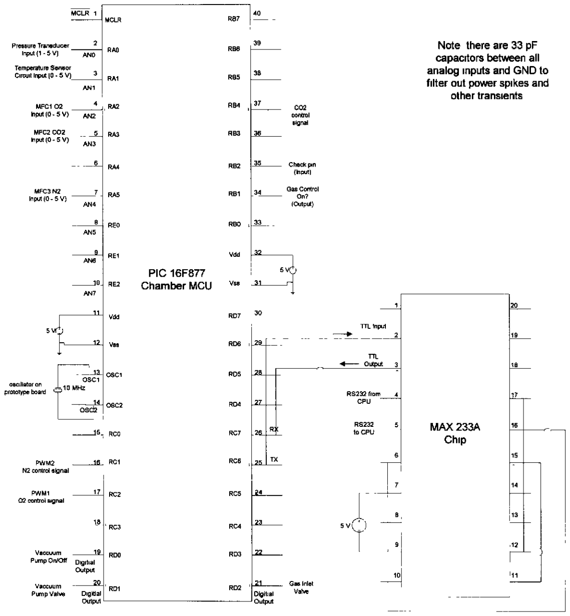
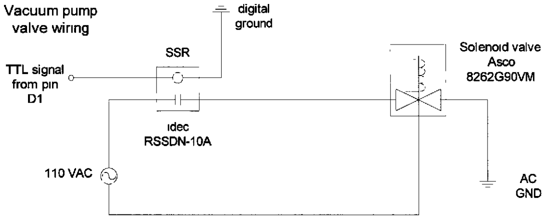
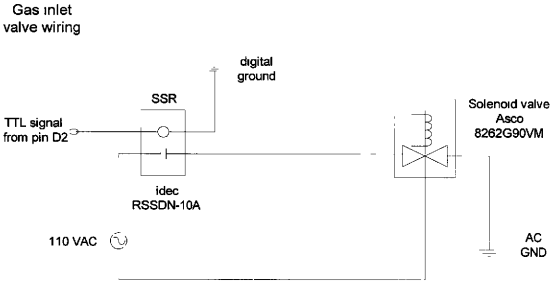


Figure B-1. Schematic of MCU and MAX233A serial conversion chip and associated wiring. All analog and digital signals are labeled.

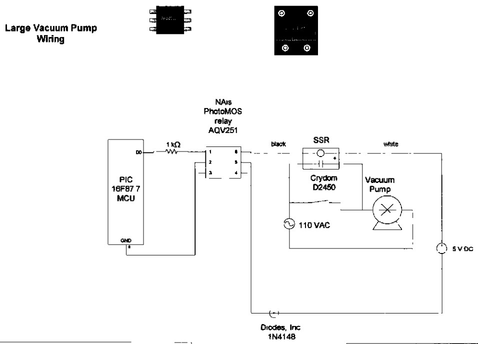


(a)



(b)

Figure B-4. Wiring schematics for solenoid valves. (a) Vacuum pump solenoid valve and associated relay. (b) Gas inlet solenoid valve and associated relay.



9 pin connector assignments inside box		
Pin	Description	Color
1	to relay - from opto pin 6	white
2	from relay +	yellow
3	to 5 V	yellow
4	from 5 V COM to opto pin 5	yellow
* pins 2 and 3 are jumpered together		

9 pin connector assignments outside box		
Pin	Description	Color
1	to relay -	black
2	from relay +	white
3	to + 5 V	black
4	from 5 V COM	white

Figure B-5. Schematic of relay wiring for vacuum pump. Two relays are necessary to operate the vacuum pump.

APPENDIX C: SYSTEM CHECK CALCULATIONS

Setpoints	
Pressure	80
O2	200
CO2	1000
N2	799

Current	
Pressure	71 449
O2	200
CO2	1000
N2	799

Calculations	
n_gas	1 7592
n_O2	0 3518
n_CO2	0 0018
n_N2	1 4056
n_gas_set[0]	1 9698
n_gas_set[1]	1 7592
n_gas_set[2]	1 7592
n_gas_set[3]	1 7592
n_gas_pivot	1 9698
n_add[1]	0 0421
n_add[2]	0 0002
n_add[3]	0 1682
v_add[1]	0 9423
v_add[2]	0 0047
v_add[3]	3 7697

10-bit to kPa conversion
494 10-bit
71 44882209 kPa

Constants	
V_chamber	60 4
R	8 314
T	21 9
rho_O2	1 43
rho_CO2	1 98
rho_N2	1 25
MW_O2	32
MW_CO2	44 01
MW_N2	28 01

Trial Data - Pressure Change

Pressures		CPU values	MCU values	% composition	% dif
50	v_add[1]	1 01984	1 067457	19 98%	-4 67%
60	v_add[2]	0 005473	0 005314	0 10%	2 91%
	v_add[3]	4 408402	4 270152	79 92%	3 14%
60 1	v_add[1]	1 090964	1 084101	19 98%	0 63%
70	v_add[2]	0 005418	0 005384	0 10%	0 63%
	v_add[3]	4 364318	4 336862	79 92%	0 63%
70	v_add[1]	1 01984	1 074315	19 98%	-5 34%
80	v_add[2]	0 005473	0 005335	0 10%	2 52%
	v_add[3]	4 408402	4 297731	79 92%	2 51%
55 1	v_add[1]	0 539972	0 538188	19 98%	0 33%
60	v_add[2]	0 002682	0 002672	0 10%	0 37%
	v_add[3]	2 160117	2 15298	79 92%	0 33%
60 1	v_add[1]	0 319575	0 26005	19 98%	18 63%
63	v_add[2]	0 001587	0 001291	0 10%	18 65%
	v_add[3]	1 278437	1 040312	79 92%	18 63%
63 2	v_add[1]	0 308555	0 299453	19 98%	2 95%
66	v_add[2]	0 001532	0 001487	0 10%	2 94%
	v_add[3]	1 234353	1 19794	79 92%	2 95%
66 4	v_add[1]	0 396714	0 422556	19 98%	-6 51%
70	v_add[2]	0 00197	0 002098	0 10%	-6 50%
	v_add[3]	1 587025	1 690404	79 92%	-6 51%

NEW PROGRAM

60 1	v_add[1]	1 09096403	0 92533	19 98%	15 18%
70	v_add[2]	0 00541817	0 004595	0 10%	15 19%
	v_add[3]	4 36431782	3 701713	79 92%	15 18%
70 2	v_add[1]	1 07994419	1 074315	19 98%	0 52%
80	v_add[2]	0 00536344	0 005335	0 10%	0 53%
	v_add[3]	4 3202338	4 297718	79 92%	0 52%
70	v_add[1]	1 07994419	0 942007	19 98%	12 77%
80	v_add[2]	0 00536344	0 004678	0 10%	12 78%
	v_add[3]	4 3202338	3 768427	79 92%	12 77%
71 4	v_add[1]	0 88938446	0 889083	19 98%	0 03%
80	v_add[2]	0 00441704	0 004415	0 10%	0 05%
	v_add[3]	3 55791425	3 55671	79 92%	0 03%
71 45	v_add[1]	0 94232601	0 942007	19 98%	0 03%
80	v_add[2]	0 00467997	0 004678	0 10%	0 04%
	v_add[3]	3 76970284	3 768427	79 92%	0 03%

Figure C-1. Spreadsheet used to calculate the amount of each gas to added to the chamber. This value was compared to the values of gas added returned by the MCU. As the pressure reading became more accurate and more precise, the difference between the two values for each gas decreased.

Gas Addition Calculation Worksheet

LabVIEW Setpoints

Setpoints		Current Conditions	
Pressure	101 3 kPa	Pressure	101 3 kPa
O2	21 %	O2	21 %
CO2	400 ppm	CO2	0 13 %
N2	78 96 %	N2	78 %

MCU Setpoints

Setpoints		Current Conditions	
Pressure	101 3 kPa	Pressure	101 3 kPa
O2	210 ppt	O2	210 ppt
CO2	400 ppm	CO2	1300 ppm
N2	789 6 ppt	N2	780 ppt

Constants

V_chamber	55 3 L
R	8.314 L-kPa/mol-
rho_O2	1.43 g/L
rho_CO2	1.98 g/L
rho_N2	1.25 g/L
MW_O2	32 g/gmol
MW_CO2	44.01 g/gmol
MW_N2	28 01 g/gmol
Temperature	25 C

n_gas	2 259903 mols
n_O2	0 47458 mols
n_CO2	0 002938 mols
n_N2	1 762724 mols
n_gas_set[0]	2 259903 mols
n_gas_set[1]	2 259903 mols
n_gas_set[2]	7.344684 mols
n_gas_set[3]	2.232427 mols
n_pivot_gas	7.344684 mols
n_add[1]	1.067804 mols
n_add[2]	0 mols
n_add[3]	4.036638 mols
v_add[1]	23.89492 L
v_add[2]	0 L
v_add[3]	90 45299 L

Final Pressure	330 1063 kPa

n_total	7.364345 mols

add_time[1]	23 89492 min
add_time[2]	0 min
add_time[3]	90 45299 min
total	114.3479 min
	1 905798 hr

Figure C-2. Spreadsheet used to determine the correct amount of each gas to add to the chamber based on current conditions and setpoints. This spreadsheet was used to determine the total number of moles of gas to add, the time it would take to add the gas at the appropriate flowrate, and the final pressure after gas addition.

VITA

Denise Lynn Brown
P.O. Box 129
Tilden, TX 78072

The author was born in Kerrville, Texas, 1977, the first daughter of Alan and Gail Brown. She has a younger brother, Justin, and a younger sister, Linda. The author graduated from McMullen County High School in 1995. She continued her education at Texas A&M University in College Station, Texas and graduated with a Bachelor of Science degree in Agricultural Engineering. She began graduate school at Texas A&M in December 1999. While in graduate school, the author worked as both a teaching assistant and a research assistant in the Department of Biological and Agricultural Engineering. In her last semester, she took a position in the department as an assistant lecturer for a senior level design class in controlled environments. While attending Texas A&M University, she was a member of Phi Eta Sigma National Honor Society and the Golden Key National Honor Society. She also competed on the Equestrian Team in the Intercollegiate Horse Show Association, and competed internationally as a member of the Silver Spurs of Aggieland, a country and western dance team attending United Country Western Dance Council and Fun Country events. She played intramural volleyball and tennis and played volleyball in the College Station City League. The author's hobbies include reading, dancing, riding horses, rollerblading, volleyball, computers, sailing, motorcycle riding, and much more.